AFRL-IF-RS-TR-2006-250
Final Technical Report
July 2006

INTEGRATION OF AUDIT DATA ANALYSIS AND MINING TECHNIQUES INTO AIDE

George Mason University

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-250 has been reviewed and is approved for publication.

APPROVED: /s/

BRIAN T. SPINK
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY Jr., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUL 06	Final	Mar 00 – Dec 05

4. TITLE AND SUBTITLE
INTEGRATION OF AUDIT DATA ANALYSIS AND MINING TECHNIQUES INTO AIDE

5a. CONTRACT NUMBER
F30602-00-2-0512

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)
Sushil Jajodia

5d. PROJECT NUMBER
4519

5e. TASK NUMBER
32

5f. WORK UNIT NUMBER
P1

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
George Mason University
4400 University Drive
Fairfax Virginia 22030-4444

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFGA
525 Brooks Rd
Rome New York 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-250

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA#06-526

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In recent years, intrusion detection systems have gained wide acceptance within both government and commercial organizations. A number of intrusion detection tools are commercially available and are being routinely used as part of the protection of network and computer systems. There are several limitations to the present generation of the intrusion detection systems: these tools detect only those attacks that are already known, generate too many false positives, and operation of these tools is too labor intensive. To overcome these problems, we developed methods and tools that can be used by the system security officer to understand the massive amount of data that is being collected by the intrusion detection systems, analyze the data, and determine the importance of an alarm. Report divided into three parts. Part I describes a network intrusion detection system, called Audit Data Analysis and Mining (ADAM), which employs a series of data mining techniques including association rules, classification techniques, and pseudo-Bayes estimators to detect attacks using the network audit trail data. Part II shows how to build attack scenarios by explicitly including network vulnerability/exploit relationships in the model. Part III provides a complete list of publications resulting from this effort and successfully licensed the resulting technology to a company called Secure Decisions and filed for four patents.

15. SUBJECT TERMS
Intrusion Detection, Audit Data Analysis and Mining (ADAM), pseudo-Bayes

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Brian T. Spink
U	U	U	UL	50	19b. TELEPONE NUMBER (*Include area code*)

Abstract

In recent years, intrusion detection systems have gained wide acceptance within both government and commercial organizations. A number of intrusion detection tools are commercially available and are being routinely used as part of the protection of network and computer systems.

There are several limitations to the present generation of the intrusion detection systems: These tools detect only those attacks that are already known, generate too many false positives, and operation of these tools is too labor intensive. To overcome these problems, we developed methods and tools that can be used by the system security officer to understand the massive amount of data that is being collected by the intrusion detection systems, analyze the data, and determine the importance of an alarm.

This report is divided into three parts. Part I describes a network intrusion detection system, called Audit Data Analysis and Mining (ADAM), which employs a series of data mining techniques including association rules, classification techniques, and pseudo-Bayes estimators to detect attacks using the network audit trail data. Although association rules have been used before for this purpose, our technique is novel in two ways: First, it compares the suspicious rules with a previously generated profile of "normal" rules and selects only rules not in that profile. Second, it does the mining incrementally by sliding a time window over the audit trail data, as it is being generated. Experiments using the DARPA Intrusion Detection Evaluation Data show that ADAM is very successful in detecting two types of attacks: TCP-dump probe attacks and DOS attacks, generating very few false positives per day.

In Part II, we show how to build attack scenarios by explicitly including network vulnerability/exploit relationships (i.e., the attack graph) in the model. We map intrusion events to known exploits in the network attack graph, and correlate the events through the corresponding attack graph distances. From this, we construct attack scenarios, and provide scores for the degree of causal correlation between their constituent events, as well as an overall relevancy score for each scenario. While intrusion event correlation and attack scenario construction have been previously studied, this is the first treatment based on association with network attack graphs. We handle missed detections through the analysis of network vulnerability dependencies, unlike previous approaches that infer hypothetical attacks. In particular, we quantify lack of knowledge through attack graph distance. We show that low-pass signal filtering of event correlation sequences improves results in the face of erroneous detections. We also show how a correlation threshold can be applied for creating strongly correlated attack scenarios. Our model is highly efficient, with attack graphs and their exploit distances being computed offline. Online event processing requires only a database lookup and a small number of arithmetic operations, making the approach feasible for real-time applications.

In part III, we provide a complete list of publications that have resulted from this effort. Moreover, we successfully licensed the resulting technology to a company called Secure Decisions and filed for four patents. We have acknowledged AFRL/Rome support in the patent applications.

Table of Contents

Part I List of Figures

Part II List of Figures

Part I. ADAM: Audit Data Analysis and Mining

1 Introduction

The most recent network attack, SQL worm, resulted in massive packet loss throughout the web, caused severe latency, and caused five of the thirteen root nameserver to fail. This incidence reveals a simple fact that network-based attacks continue to increase in frequency and severity.

Intrusion detection techniques can be classified into two broad categories: misuse detection and anomaly detection. Misuse detection aims to detect well-known attacks as well as slight variations of them, by characterizing the rules that govern these attacks. Due to its nature, misuse detection systems generally have low false alarms but they are unable to detect any attacks that lie beyond their knowledge. Three major techniques are widely used in misuse detection: expert systems [10, 33, 31, 20, 32, 7], signature analysis, and state transition analysis [36, 41]. The other techniques include data mining [26, 29, 28, 27] and petri net [9].

Anomaly detection is designed to capture any deviations from the established profiles of users and systems normal behavior pattern. Profiles of normal behavior can be built with a variety of techniques including statistical [10, 37], association rule [5], neural network [17], computer immunology [14], and specification-based [38] methods. In principle, anomaly detection has the ability to detect new attacks. In practice, this is far from easy because of two reasons. First, it is very hard to obtain accurate and comprehensive profiles of the normal behavior, which makes an anomaly detection system easily generate many false alarms. Second, it is hard to discriminate between normal deviations and abnormal deviations from the profiles, which makes it difficult for an anomaly system to distinguish the true intrusions from the normal instances even if new attacks are captured by the system. Thus, anomaly detection has the potential to generate too many false alarms, and often it can be very time consuming and labor expensive to sift true intrusions from the false alarms.

There are several limitations to the present generation of the intrusion detection systems. These tools are only effective in detecting attacks that are already known; they often generate too many false positives; operation of these tools is too labor intensive. To overcome these problems, we need to develop methods and tools that can be used by the system security officer to understand the massive amount of data that is being collected by the intrusion detection systems, analyze the data, and determine the importance of an alarm.

The use of specialized audit trails for intrusion detection has been advocated by security experts as a means to discover previously unknown attacks. The idea is to analyze the audit trail to spot "abnormal" patterns of usage, performing intrusion detection; however, audit trails contain large amounts of data, making the task of intrusion detection very laborious. In fact, in order not to be bypassed by potential intruders, it is advisable to collect data at the lowest possible level (e.g., monitoring system service calls as opposed to application-level monitoring). On the other hand, the lower one pushes the monitoring, the larger the size of the data collected. To alleviate this problem, the use of random sampling has been suggested; however, using sampling one runs the risk of missing intrusions.

The problem is further complicated by the need to allow for differences in the data due to special circumstances such as holidays and other factors. For instance, the "normal" number and duration of FTP connections may vary from morning to afternoon to evening. It may also depend on the day of the month or the week, or it may vary depending on the class of users being considered.

In this report, we describe a network intrusion detection system, called ADAM, which applies a set of data mining techniques including association rules, classifications, and pseudo-Bayes estimators to detect network intrusion. Even though mining of association rules has been used to detect intrusions in audit trail data [26, 29], ADAM is unique in two ways:

- ADAM uses incremental mining (on-line mining): It does not look at a batch of TCP connections, but rather uses a sliding window of time to find the suspicious rules within that window.
- ADAM is profile-based: It builds, a-priori, a profile of "normal" rules, obtained by mining past periods of time in which there were no attacks. Any rule discovered during the on-line mining that belongs to this profile is ignored, assuming it corresponds to a normal behavior. In this sense, ADAM looks for unexpected rules. This helps in reducing the number of false positives flagged by the technique.

Additionally, ADAM has two distinct properties:

- It is able to detect novel attacks. As we stated earlier, detecting new attacks is one of the hardest tasks to accomplish, since no knowledge about these attacks is available. ADAM uses pseudo-Bayes estimators to enhance the ability to detect new attacks [6].
- It builds profiles using temporal association rules in terms of multiple time granularities [30]. This provides more flexibility in defining time intervals, and thus produces more precise profile about the temporal patterns of user and system behavior.

In addition, ADAM gives a security operator the capability of drilling down into the audit trail data in order to examine the origin of the suspicious activity, facilitating the human intervention in the analysis by keeping the focus of the operator on the subset of data that is probably causing the attack.

The rest of the report is organized as follows. Section 2 gives a brief review of the related work, followed by the data mining applications in intrusion detection. Sections 3 and 4 present the framework and techniques used in ADAM. Section 5 descibes the experimental results based on the DARPA data, and Section 6 offers the summary and conclusions of this research.

2 Related Work

ADAM can be cataloged as using a profile base technique to identify expected (an unexpected) behavior. Profiles, based on models of observed activity of subjects were introduced as a tool for intrusion detection by D. Denning [11]. The model used by Denning is a rule-based pattern

matching system, independent of the target system for which it is detecting intrusions. Audit records, with no standard format provided, are collected and analyzed. Each observation in the trail is used to form the profiles. Then, various statistical models are used to determine the current activity model and look for intrusions. Our technique uses the same principle, although we focus in the discovery of abnormal association rules instead of arbitrary patterns.

The IDES prototype, described in [10, 33], was developed in SRI and it is based on the statistical and expert rule-based techniques. The intrusion detection measures are selected based on the intuition and experience of the developers' group. NIDES [3] extended the work of IDES by introducing a results-fusion component to integrate its response logic with the results produced by the anomaly-detection subsystem. A successor system to NIDES, called EMERALD [35], currently under development at SRI, extends NIDES to accommodate network analysis.

NetSTAT [41] is a tool aimed at real-time network-based intrusion detection, that uses state transition analysis to represent attack scenarios in a networked environment.

eBayes of SRI's Emerald uses Bayes net technology to analyze bursts of traffic [40]. Defining a session as temporally contiguous bursts of TCP/IP traffic from a given IP, eBayes applies Bayesian inference (based on naive Bayes model) on observed and derived variables of the session, to obtain a belief for the session over the states of hypothesis. Hypothesis can be normal events and attacks. Given a naive Bayes model, a training data, and a set of hypothesis, a conditional probability table(CPT) is built for the hypothesis and variables, and it will be adjusted for the current observations. By adding a dummy state of hypothesis and a new CPT row initialized by a uniform distribution, eBayes has ability to generate the new hypothesis dynamically that helps it to detect new attacks. DuMouchel proposes a statistical method that compares the sequence of each user's commands to a stored profile describing the probability distribution of that user's command sequences [13]. It uses a Bayes Factor statistic to test the null hypothesis that the observed command transition probabilities come from a profiled transition matrix. The alternative hypothesis is formed as a a Dirichlet mixture of multinomial command probabilities.

Forrest et al. record frequent subsequences of system call that are invoked in the execution of a program [14]. Absence of subsequences in the current execution of the same program from the stored sequences constitutes a potential anomaly. Lane and Brodley use a similar approach but they focus on an incremental algorithm that updates the stored sequences and used data from UNIX shell commands [25]. Using a technique based on Teiresias algorithm, Wespi et al. extend Forrest's approach of modeling processes by fixed-length sequences of system calls to variable-length patterns, which are more naturally suited to present the process model [42]. Lee et al. using a rule learning program, generate rules that predict the current system call based on a window of previous system calls. Abnormality is suspected when the predicted system call deviates from the actual system call [28]. Ghosh and Schwartzbard propose using a neural network to learn a profile of normality [17].

2.1 Intrusion Detection using data mining techniques

Data mining refers to a process of nontrivial extraction of implicit, unknown, and potential useful information from databases. There are other terms that carry a similar meaning to data mining, such as knowledge discovery in databases, knowledge extraction, and data/pattern analysis.

In the past several years, data mining techniques have attracted a lot of attention in the area of intrusion detection because of two reasons:

- Many data mining techniques are well suited for the needs of intrusion detection. Examples of the techniques used in intrusion detection systems include decision tree, link analysis, association rules, clustering, rule abduction, and sequence analysis.

- From the data analysis point of view, the task of intrusion detection is closely related to data mining. An intrusion detection system aims at finding attack activity from usually very large volume of audit trails data, which is infeasible to be handled manually. The entire process of intrusion detection is generally involved in some of the following operations: audit trail data preprocessing, audit data analysis, pattern extraction for attacks, "normal" activity or both, and so on.

Several intrusion detection systems have been developed based on the data mining techniques (see, for example, [24,3,1]). JAM is a misuse detection system developed at Columbia University. It employs mining association rules and frequent episodes to discover patterns of intrusions, then uses a meta-learning classifier to learn the signature of attacks [26, 29, 28, 27]. The association rules algorithm is used to determine relationships between fields in the audit trail records. The frequent episode algorithm is used to model sequential patterns of audit events. Then features are extracted from both algorithms and used to compute the models of intrusion behavior. The meta-learning classifiers build the signature of attacks. ADAM is a network anomaly detection system [6, 5]. It employs an association rule mining module to learn system profile and to capture suspicious patterns of network traffic. A classification module is used to learn the output of mining module so as to further reduce the false alarms and assign right names to known attacks. IDDM characterizes change between network data descriptions at different times, and produces alarms when detecting large deviations between descriptions [1]. Helmer et al. present a distributed intrusion detection system that uses data mining agents to automate discovery of concise rules from system call traces [19]. The intrusion detection agents are deployed at two levels. Low level agents travel to each monitored component, gather recent information of network, and classify the data to determine whether suspicious activity is occurring. High-level agents maintain the data warehouse by combining knowledge and data from the low-level agents. A set of machine learning algorithms are used to discover patterns of coordinated intrusions. Bala et al. propose an approach to building a network profile by applying distributed data analysis methods [4]. Global profiles are built using a Distributed Data Mining approach that integrates inductive generalization and Agent based computing. Agents generate partial trees and communicate the temporary results among them in the form of indices to the data records. Then the classification rules are learned via tree induction from distributed data to be used as intrusion profiles.

Data mining techniques have also been used in virus detection and alarm analysis. Shultz et al. propose a framework that uses data mining algorithms based on RIPPER, Naive Bayes and multi-Naive Bayes to train multiple classifiers on a set of malicious and benign executables to detect new malicious binaries [39]. S. Manganaris et al. propose to use association rules to detect anomaly on the IDS alarm stream [34]. Julish et al. propose an alarm clustering algorithm to manage intrusion detection alarms by identifying and resolving their root causes [22], and a conceptual clustering technique to mine historical alarms so as to handle future alarms more effectively [23]. Klemettinen designed an alarm correlation system that uses association rules and frequent episodes algorithms to discover alarm patterns [24].

3 ADAM System

3.1 Architecture of ADAM

ADAM is a network anomaly detection system, and its architecture is shown in Figure 1. It is composed of two modules: Preprocessing Engine and Mining Engine. The Preprocessing Engine sniffs TCP/IP traffic data, extracts header information of each packet, and generates a record for each connection based on a predefined schema

$$R = \{T_s, Src.IP, Src.Port, Dst.IP, Dst.Port, Flag\}$$

where T_s gives the starting time of a connection, $Src.IP$ and $Src.Port$ are source IP and port of a connection, and $Dst.IP$ and $Dst.Port$ are destination IP and port. Flag gives the connection status, which can be open, half open, close, etc. ADAM only works on TCP/IP header information of the TCP/IP packets, and it does not analyze the payload of them.

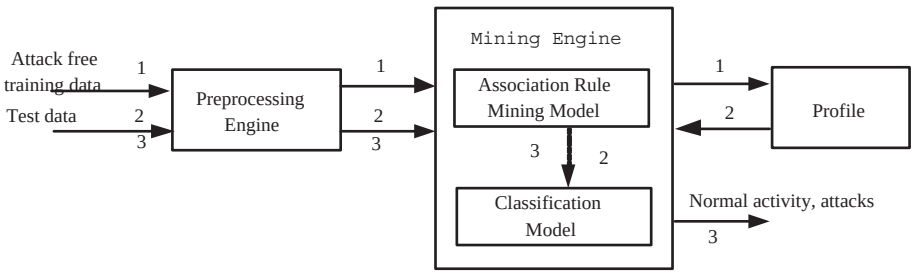

Figure 1: Frame work of ADAM

The Mining Engine contains two components: Association Rule Mining module and Classification module. The Association Rule Mining module searches for the high support association rules of the connection records. It contains three association rule algorithms: single-level mining, domain-level mining and feature selection. The Classification module contains a set of classification algorithms. It accepts the association rules from the mining module and classifies them into attacks and normal activity. Normal activity is discarded and only attacks are presented to the system security officer.

3.2 Operation of ADAM

ADAM performs intrusion detection task in two phases: training phase (denoted by 1, 2 in Figure 1) and detecting phase (denoted by 3 in Figure 1). Given a training data with attacks labeled, the training phase is composed of two stages. At the first stage, the Association Rule Mining module searches for the high support association rules of an attack free data obtained by removing all the attack connections from the training data, and puts them into the profile, because they are believed to represent normal behavior patterns. At the second stage, ADAM takes the whole training data and searches for high support association rules. Then it compares them with the profile, and outputs the suspicious rules, that are either not in the profiles or are very different from the profiles. The suspicious association rules are converted into the classification vectors, and train the Classification module. In the detecting phase, Association Rule Mining module searches for the suspicious association rules of the audit data, and translated them into the vectors. Then the Classification module classifies them into normal events and attacks. The normal events are ignored, and only attacks are output to the system security officer.

4 Techniques

ADAM uses a combination of on-line mining of *association rules* with *classification* algorithms to identify and characterize suspicious activity in the audit trail.

4.1 Mining association rules

The task of mining association rules, first presented in [2] consists in deriving a set of rules in the form of $X \longrightarrow Y$ where X and Y are sets of attribute-values, with $X \bigcap Y = \emptyset$ and $\|Y\| = 1$. The set X is called the antecedent of the rule while the item Y is called consequent. For example, in a market-basket data of supermarket transactions, one may find that customers who buy *milk* also buy *honey* in the same transaction, generating the rule $milk \longrightarrow honey$. There are two parameters associated with a rule: *support* and *confidence*. The rule $X \longrightarrow Y$ has *support* s in the transaction set T if $s\%$ of transactions in T contain $X \cup Y$. The rule $X \longrightarrow Y$ has *confidence* c if $c\%$ of transactions in T that contain X also contain Y. The most difficult and dominating part of an association rules discovery algorithm is to find the itemsets $X \bigcup Y$ that have strong support. (Once an itemset is deemed to have strong support, it is an easy task to decide which item in the itemset can be the consequent by using the confidence threshold.) For this reason, we actually aim to find high-support itemsets in our technique, rather than their rules. We use the terms rules and itemsets interchangeable throughout the report.

Association Rule Mining module contains three association rule mining algorithms: *single-level mining*, *domain-level mining*,[1] and *feature selection*. Both *single-level mining* and *domain-level mining* algorithms can work in two modes: static mode and dynamic mode. In static mode,

[1]This is known as multi-level mining in data mining terminology; to avoid confusion with multi-level security concept, we use domain-level instead in this report

$Src.IP, Dst.IP, Serv, Dur$
$Src.IP, Dst.Port, Serv, Dur$
$Src.IP, Dst.IP, Dst.Port, Serv, Dur$
$Src.IP, Src.Port, Dst.IP, Serv, Dur$
$Src.IP, Src.Port, Dst.Port, Serv, Dur$
$Src.IP, Src.Port, Dst.IP, Dst.Port, Serv, Dur$

(a) single-level mining

$Src.IP, Dst.Sub$
$Src.IP, Dst.Sub, Dst.Port$
$Src.Sub, Dst.IP$
$Src.Sub, Dst.IP, Dst.Port$
$Src.Sub, Dst.Sub$
$Src.Sub, Dst.Sub, Dst.Port$

(b) domain-level mining

Figure 2: Rule set of single-level mining and domain-level mining

the algorithm scans the entire database and generates the "global" frequent association rules. In dynamic mode, the algorithm applies a sliding window on the database and generates the frequent association rules for the records in each window. For simplicity, we only present the dynamic algorithms in the following discussion.

4.1.1 Single-level mining

The relation R contains the dataset that is subject of the association mining. Notice that in this context, the association rules we can come up with are more restrictive than in the general market-basket data case. For instance, there can be no rules with two different $Src.IP$ values in the antecedent (No single connection comes from two sources). Nevertheless, the number of potential rules is large: connections may come from a large base of source IP addresses and ports. Let $S = \{Src.IP, Src.Port\}$, $D = \{Dst.IP, Dst.Port\}$. We are interested in itemsets that are in the form of $s \rightarrow d$, where $s \in S \wedge\ Src.IP \in s$ and $d \in D$. Figure 2(a) shows the single-level itemsets.

Notice that we have deliberately omitted itemsets that only contain $Src.Prt$ without containing $Src.IP$, since we think that these sets lack the valuable correlation between source and destination hosts.

The dynamic single-level mining algorithm implements incremental on-line association rule mining. It only needs to look at each connection record once, thus it is appropriate for on-line detection. The sliding window size is denoted as Δw. This window size can be interpreted as either counting the last Δw connections, or a the last Δw units of time. Let P_{c_h} denote a pointer to the first connection c_h in a window, and c the current record being processed in the window. Let $c(P_{c_h})$ define an operation such that $c_h = c(P_{c_h})$, and $\prod_i(c)$ be the projection of c to the corresponding type of association rule shown as in Figure 2(a).

Figure 3 shows the pseudo-code of dynamic single-level mining. Notice that, at the first stage of training phase the algorithm generates the profile of system since the input dataset is attack free. In this case, $H_p^i = \emptyset$ for $i = 1, \ldots, 6$, and the output L_i of the algorithm for $i = 1, \ldots, 6$ make the profile. In the test phase, L_i gives a set of suspicious association rules associated with $\prod_i$.

The dynamic single-level mining algorithm proceeds as follows. First it takes the current connection c recorded in the audit trail (corresponding to the preprocessing of TCP packets) and examines if any of its projections $\prod_i(c)$ is in the corresponding table H_p^i. If this is not the case, it inserts the projection into the table H_i. Otherwise, the projection is ignored, since it is a normal occurrence.

Input: *H_p^i (i=1,...,6): the hash tables of normal association rules, winSize, sup. Each H_p^i stores one of six types of associations*
Output: **L***: A set of new association rules*

begin

(1) **for all** *connection records c* **do begin**
(2) **if** c is the first record **then**
(3) $p_h = p_c$
(4) $c_h = c$
(5) $t_h = c.Ts$
(6) **if** $c.Ts - t_h <= winSize$ **then**
(7) **for** $i = 1, \ldots, 6$
(8) HashUpdate$(H^i, H_p^i, \prod_i(c))$
(9) **else**
(10) **for** $i = 1, \ldots, 6$
(11) $L_i = L_i \cup$ HFilter(H^i, sup)
(12) **while** $c.Ts - t_h > winSize$
(13) $c_h = r(p_h)$
(14) $t_h = C_h.Ts$
(15) $p_h = p_h + 1$
(16) HDelete$(H^i, \prod_i(C_h))$
(17) **return** $L_1, \ldots, L_6$

end

HashUpdate$(H^i, H_p^i, \prod_i(c))$ {
if $(\prod_i(c) \notin H_i$ and $\prod_i(c) \notin H_i^p)$
inserts$\prod_i(c)$ into H^i;
if $(\prod_i(c) \notin H_i^p$ and $\prod_i(c) \in H^i)$
$\prod_i(c).sup = \prod_i(c).sup + 1$;
}
HDelete$(H^i, \prod_i(c))$ {
if $\prod_i(c).sup = 1$
delete $\prod_i(c)$ from H^i
else
$\prod_i(c).sup = \prod_i(c).sup - 1$
}
HFilter(H^i, sup) {
$G_i = \emptyset$
for all entries e in H^i
if $e.sup \geq sup$
$G_i = G_i \cup e$
return G_i
}

Figure 3: Dynamic single-level mining algorithm

Starting with the first connection record c_h in the audit trail, the algorithm processes connection records and updates the hash tables H_i until it reaches a connection record c such that it cannot be in the same window with p_h, *i.e.*, $c.Ts - c_h.Ts > winSize$. Here, $c_h.Ts$ denotes the starting time of the connection c_h, and $c.Ts$ the starting time of c. Then the algorithm updates each L_i with the entries in H_i whose supports are higher than a predefined threshold *sup*. Then the algorithm slides down the window and locates the head c'_h (the first record) of the new window such that $c.Ts - c'_h.Ts \leq winSize$. Let S be the set of records that are moved outside of the new window. The algorithm updates H_i by reducing the support for the projections of each record in S. A projection is removed from H_i if its support reaches to zero. The algorithm repeats until the entire database has been processed.

4.1.2 Domain-level mining

Sometimes an attack will take place as a coordinated effort from several hosts or to multiple hosts, such as distributed denial of service attack (DDOS), Internet worm, etc. In that case, it is likely that itemsets of the form $\{Src.IP, Dst.IP\}$ will not have enough support to be flagged as suspicious. However, if we aggregate the support of all the itemsets of the same form where all the source IPs or destination IPs belong to the same subnet, the support may be enough to recognize this as a suspicious event. So we complemented our technique by using domain-level mining, which roughly speaking, can be thought of clustering itemsets that have some commonality and aggregating the support that the individual (flat) itemsets exhibit. The mining of multi-level association rules were first proposed by J. Han et al. [18]. We use a bottom-up method to produce the domain-level rules.

Given the schema R:

$$R(T_s, Src.IP, Src.Port, Dst.IP, Dst.Port, FLAG)$$

we can produce higher abstractions of the IP related attributes. For instance, the first byte of the IP address usually identifies the subnet to which the host belongs, while the first two bytes of the IP address identify the net ID. In the domain level mining, we define four layered subnets from the lowest level to highest level: Sub_1 which is identified by the first three bytes of the IP address, Sub_2 by the first two bytes of the IP address, Sub_3 by the first 1 byte of the IP address, Sub_4 is the highest level subnet that contains all possible IPs. Clearly Sub_1 is the first level abstraction on IP address, Sub_2 is the first level abstraction on Sub_1, and so on. This gives rise to the itemsets of interest showed in Figure 2(b), in which Sub can be either one of the four subnets. For instance, the itemset $\{Src.IP, Dst.Sub\}$ represents the association between a source IP and a destination subnet, and it includes four possible types of associations: $\{Src.IP, sub_1\}$,$\{Src.IP, sub_2\}$,$\{Src.IP, sub_3\}$,$\{Src.IP, sub_4\}$. Similarly, $\{Src.Sub, Dst.IP\}$ consists of four types of associations, and $Src.Sub, Dst.Sub$ consists of $4^2 = 16$ possible types of associations. Figure 4 shows all possible domain-level associations.

The dynamic domain-level mining algorithm is very similar to the single-level mining algorithm, except that it needs more hash tables to store all possible domain-level association rules. For brevity, we do not show the pseudo-code of domain-level algorithm.

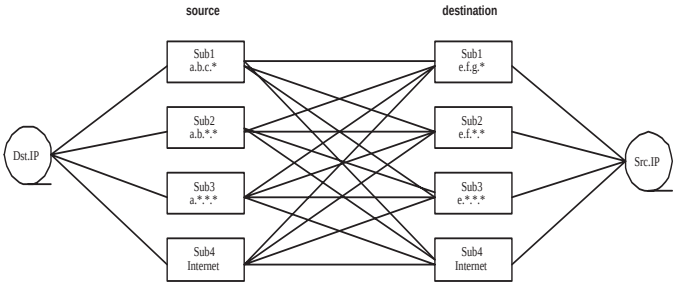

Figure 4: Domain-level association rules

4.1.3 Feature selection

Feature selection by nature is a multi-windows mining process. Both single-level mining and domain-level mining algorithms use single size time window, and it is very hard to choose an optimum window size that can capture all kinds of rules that appear at different frequencies. For instance, if the time window is too big, the algorithms may miss some rules that are hot only in a short period of time. On the other hand, if the time window is too small, they may miss the rules that span a long time period in a slow manner. Our motivation to do feature selection is to overcome the limitations of single window size mining algorithms. Two time windows are used here. One is three seconds wide, and it is used to capture the rules that are only hot in a very short period of time and can easily be missed in a wide window. The other is 24 hours long, and it is used to capture the rules that appear at very low frequency but last for long time. Both time windows are applied on *single-level mining* and *domain-level Mining* algorithms. Some features like average connection rate per second, contiguity index of a source IP to a set of destination IP, etc., are extracted from the mining results and will be used for the further analysis.

4.1.4 Profiling process by temporal association rules

Temporal association rules is one of new features being added to ADAM. They provide more flexibility in defining time intervals, and thus produce more precise profile regarding to temporal patterns of user and system behavior.

Generally, user behavior during different time intervals tend to be very different. For example the "normal" number and duration of FTP connections may vary from working hours to midnight, from business day to weekend or holiday. Furthermore, these variations may depend on the day of the month or the week. However, the time factor, expeically in terms of multiple time granualrity, has not been utilized extensively in generation of profiles by association rules. To capture the "time factors" in user and system normal behavior, we enhance the profiling process of ADAM by adding a temporal association rule mining module, which builds profiles using temporal association rules in terms of multiple time granularities [30].

A *temporal association rule* is an association rule with a calendar pattern represented as a pair (r, e), where r is an association rule and e is a calendar pattern on a calender schema R. Here, A *calendar schema* is a relational schema (in the sense of relational databases) $R = (f_n : D_n, f_{n-1} : D_{n-1}, \ldots, f_1 : D_1)$ together with a *valid* constraint (explained below). Each attribute f_i is a calendar unit name like year, month, and week etc. Each domain D_i is a finite subset of the positive integers. The constraint *valid* is a Boolean function on $D_n \times D_{n-1} \times \cdots \times D_1$ specifying which combinations of the values in $D_n \times \cdots \times D_1$ are "valid". This constraint serves two purposes. The first is to exclude the combinations that do not correspond to any time intervals due to the interaction of the calendar units. For example, we may have a calendar schema $(year : \{1995, 1996, \cdots, 1999\}, month : \{1, 2, \cdots, 12\}, day : \{1, 2, \cdots, 31\})$ with the constraint *valid* that evaluates $\langle y, m, d\rangle$ to True only if a combination gives a valid date (for example, $\langle 1995, 1, 3\rangle$ is a valid date while $\langle 1996, 2, 31\rangle$ is not). The second purpose of the *valid* constraint is to exclude the time intervals that we are not interested in. For example, if we do not want to consider the weekend days and holidays in our problem, we can let *valid* evaluate to False for all such days.

Given a calendar schema $R = (f_n : D_n, f_{n-1} : D_{n-1}, \ldots, f_1 : D_1)$, a *calendar pattern on the calendar schema* R is a tuple on R of the form $\langle d_n, d_{n-1}, \ldots, d_1\rangle$ where each d_i is in D_i or the wild-card symbol $*$. A *temporal profile over a calendar schema* is defined as a set of temporal association rules discovered from a set of timestamped transactions over the same calendar schema. Given a calendar schemar R, we extend *Apriori* algorithm to mine the temporal association rules. Details of temporal profiling can be found in [30].

4.2 Classification

The abnormal rules generated by mining association rules algorithm are intended to guide the further detection work. To filter out as many false positives as possible, we complement the technique by adding a classification step. There are two major functionalities of the Classification module: (1) reduce the false alarm; (2) recognize known attacks and assign them with right names.

Three classification algorithms are used in ADAM to capture known attacks. They are decision tree, Naive Bayes and cascading classifier. These algorithms classify association rules generated from Mining module into normal activity and attacks. Attacks will be further given a name.

The classifiers have two limitations. First, they can only recognize known attacks that appear in the training data. Thus, even novel attacks are captured by the mining module, they can be easily missed by the decision tree. Second, no training data with attacks instances may be available at all in reality. To overcome the limitation of classification algorithms, ADAM employs a novel technique called pseudo-Bayes estimators which is able to detect novel attacks and meanwhile to keep as low false positive as possible.

Association rules from mining module are converted into classification vectors that can be accepted by the Classification Module. The rule set obtained by each mining algorithm captures the different characteristics of the network traffic data, and thus is trained individually by the classification module. Total three classification models will be generated, one for each rule set. For instance, given a single-level association rule

$(Src.IP : 1.2.3.4 \rightarrow Dst.IP : 5.6.7.8, Dst.Port : 23, Serv : telnet, Dur : 15.20.49 - 15.29.03)$ $[support = 1000]$

the corresponding vector is $\{23, telnet, 494, 1000\}$, each field of which in order corresponds to the $Dst.Port$, $Serv$, Dur, $support$. The vector does not contain information about $Src.IP$ and $Dst.IP$ because the IP addresses do not convey any statistical characteristics about the network traffic.

To convert association rules into classification vectors, we extract a set of attributes from each rule set. The description of the chosen attributes is shown as follows. The first group of attributes are extracted from the rule set of single-level mining, the second from the domain-level mining, and the third from the feature selection algorithm. The attribute *support* in the first group and all attributes in the second group group are measured over a set of contiguous windows.

- First group (measured in a window)

  1. **Service Type:** service type of each connection.
  2. **Destination Port:** generally related with service type.
  3. **Duration Time:** duration time of a connection.
  4. **Support:** Number of occurrences of a connection during a time window.
  5. **Pair1: parameters from the itemset** $Src.Sub \rightarrow Dst.IP$
     - **Number of distinct destination IPs**: the first parameter in this pair measures the number of distinct source IPs in $Src.Sub$.
     - **Number of connections**: the second parameter in this pair measures the number of different connections (several connections may be from the same $Src.IP$ in $Src.Sub$) that comply with the rule.
  6. **Pair2: parameters from the itemset** $Src.IP, Dst.Sub$
     - **Number of distinct destination IPs from** $Dst.Sub$: the parameter measures the number of distinct destination IPs in $Dst.Sub$ that receive a connection from $Src.IP$.
     - **Number of connections to a destination IP from a particular subnet**: the parameter measures the number of different connections from $Src.IP$ to the particular $Dst.Sub$.
  7. **Pair3: destination subnets**
     - **Number of destination subnets receiving connections from a source IP**: Given an itemset $\{Src.IP, Dst.Sub\}$, the $Dst.Sub$ has four abstraction levels. If an itemset $\{Src.IP, Dst.Sub_i\}$ has the support higher than a predefined threshold sup, then the itemset $\{Src.IP, Dst.Sub_j\}$ has the support higher than sup if $Dst.Sub_j$ is a higher abstraction of $Dst.Sub_i$. For simplicity, the subnect is defined as the lowest abstraction level of $Dst.Sub$ such that $\{Src.IP, Dst.Sub\}$ has the support higher than a predefined threshold. So are the subnets in the following discussion.

- **Number of destination subnets receiving connections from a source subnet**: the parameter counts the number of distinct destination subnets receiving connections from a specific source subnet.

8. **Pair4: source subnets**
   - **Number of source subnets having connections to a destination IP**: the parameter counts the number of distinct source subnets having connections to a single destination IP.
   - **Number of source subnets having connections to a destination subnet**: the parameter counts the number of distinct source subnets having connections to a single destination subnect.

- Second group: measured over several contiguous windows (e.g. a day of data).

1. **Number of connections per second**.
2. **Maximum number of ports that a source IP connects to**: the maximum taken over all the windows where the itemset is flagged.
3. **Contiguity index of ports accessed from a source IP**: this parameter measures the "contiguity" of the destination ports touched by a single source IP, revealing a scanning of ports from the source IP. It is computed as the number of "holes" in the sequence of ports touched by the source IP divided by the total number of ports that received connections. For instance, if connections were found to ports 1, 3, 4 and 5, the number of holes is 1 (missing value between 1 and 3) and the total number of ports is 4, given an index of 0.25.
4. **Number of priority ports:** this counts the number of destination ports accessed that are well known ports (those from 0 through 1023).
5. **Contiguity index of IPs accessed from a source IP**: this parameter measures the "contiguity" of the destination IPs touched by a single source IP, revealing a scanning of IPs from the source IP. It is computed as the number of "holes" in the sequence of IPs where the source established connections divided by the total number of IPs that received connections.

4.2.1 Decision tree

A decision tree algorithm generates a classifier from a training data in the form of a tree structure that is either:

- a leaf, indicating a class or
- a decision node that specifies some test to be carried out on a single attribute value, with one branch and subtree for each possible outcome of the test.

A decision tree is used to classify a case by starting at the root of the tree and moving through it until a leaf is encountered. C4.5 algorithm is trained from the training data and used to classify the rules that come out of the on-line mining algorithm.

4.2.2 Naive Bayes

Naive Bayes [21, 12] relies on an assumption that given a predicted value, the attributes used for deriving the prediction are independent of each other. The idea behind Naive Bayes is as follows. Given an example $E = x_1, x_2, \ldots, x_m$ where E is composed of m attributes, Naive Bayes assigns E to the class c_k with the highest conditional probability $P(c_k|E)$. The independent assumption makes

$$\begin{aligned} P(c_k|E) &= \frac{P(E|c_k)P(c_k)}{P(E)} \\ &= \frac{P(x_1|c_k)P(x_2|c_k)\ldots P(x_m|c_k)P(c_k)}{P(E)} \end{aligned}$$

For a discrete variable x_i, $P(x_i|c_k)$ can be simply estimated by the sample frequency of training data, i.e., $P(x_i|c_k) = P(x_i, c_k)/P(c_k)$. For a continuous variable, $P(x_i|c_k)$ can be computed by the probability density function of the distribution of X_i. There are many simplified ways to estimate the *pdf*, for the simplicity we don't address here.

4.2.3 Cascading classifier

Cascading classifier sequentially runs a set of classifiers, at each step performing an extension of the original data set by adding new attributes [15, 16]. Given a set of component classifiers, cascading classifier aims at improving the overall performance by combining the decision of each component classifier.

Given a training set L, a test set T, two classifiers $\Im_1$, $\Im_2$, and a constructive operator $\Phi(D, C)$ that represents the operation to concatenate the input vectors of a dataset D with the output of classifier C, Cascading classifier works as follows:

$$\begin{aligned} Level_1train &= \Phi(L, \Im_1(x, L)) \\ Level_1test &= \Phi(T, \Im_1(x, L)) \end{aligned}$$

Classifier $\Im_2$ learns on $Level_1$ training data and classifies the $Level_1$ test data:

$$\Im_2(x, Level_1train) \quad for\ each\ x \in Level_1test$$

Classification module uses Naive Bayes at $Level_1$ and $C4.5$ at $Level_2$ to build a cascading classifier. Cascading classifier is trained along with decision tree and Naive Bayes in the training phase. In the detection phase, the final decision about a test vector is given by the cascading classifier.

4.2.4 Pseudo-Bayes estimators

Decision tree, Naive Bayes, and cascading classifier are all supervised learning algorithms. They help reduce false alarms generated by association rules algorithms, and recognize known attacks in the detecting phase. However, they require to be trained using training data that have been previously collected and in which the attacks and attack-free periods are correctly labeled. This reveals two problems with classification algorithms. First, no training data with attack instances may be available at all. Second, it is impossible to train the classifier for new forms of attacks. To overcome the limitation of classification algorithms, ADAM employs a novel technique called pseudo-Bayes estimators which is able to detect novel attacks and meanwhile to keep as low false positive as possible. Pseudo-Bayes estimators is also a new feature of ADAM.

Pseudo-Bayes estimators is a non-parametric technique of discrete multivariate analysis to provide the estimated cell values of contingency tables which may contain a large number of sampling zeros. The idea behind pseudo-Bayes is as follows [8]:

Let $\mathbf{X} = (X_1, \ldots, X_t)$ have the multinomial distribution with parameters $N = \sum_{i=1}^{t} X_i$ and $\mathbf{p} = (p_1, \ldots, p_t)$. We observe a vector of values $\mathbf{x} = (x_1, \ldots, x_t)$, where x_i is the observed count in the ith category and $\sum_{i=1}^{t} x_i = N$. The vector $\mathbf{p}$ takes values in the parameter space

$$L_t = \mathbf{p} = (p_1, \ldots, p_t) : p_t \geq 0 \qquad and \qquad \sum_{i=1}^{t} p_t = 1 \tag{1}$$

The kernel of the likehood function for this multinomial distribution is

$$l(\mathbf{p} \mid \mathbf{x}) = l(p_1, \ldots, p_t \mid x_1, \ldots, x_t) = \prod_{i=1}^{t} p_i^{x_i}. \tag{2}$$

It can be proved that both the prior and posterior distribution for this likelihood $l(\mathbf{p} \mid \mathbf{x})$ are the Dirichlet, and the the prior and posterior means of p_i are given by

$$E(p_i \mid K, \lambda) = \lambda_i \qquad (prior\ mean) \tag{3}$$

$$E(p_i \mid K, \lambda, \mathbf{x}) = \frac{x_i + K\lambda_i}{N + K} \qquad (posterior\ mean) \tag{4}$$

where

$$K = \sum_{i=1}^{t} \beta_i, \qquad \lambda_i = \frac{\beta_i}{K} \tag{5}$$

and β_i, $i = 1, \ldots, t$ are the parameters of the prior distribution of $l(\mathbf{p} \mid \mathbf{x})$.

The posterior mean is the Bayesian point estimate of $\mathbf{p}$. It can be rewritten in vector notation as

$$E(\mathbf{p} \mid K, \lambda, \mathbf{x}) = \frac{N}{N + K}(\mathbf{x}/N) + \frac{K}{N + K}\lambda \tag{6}$$

We can define risk function as the expected value of the squared distance from an estimator $\mathbf{T}$ to $\mathbf{p}$ as follows:

$$R(\mathbf{T}, \mathbf{p}) = NE\|\mathbf{T} - \mathbf{p}\|^2 = N\sum_{i=1}^{t} E(T_i - p_i)^2 \tag{7}$$

If λ is regarded as fixed, then we can find the value of K that minimizes the risk $R(\hat{\mathbf{q}}(K, \lambda), \mathbf{p})$ by differentiating (7) in K and solving the resulting equation. This yields

$$K = K(\mathbf{p}, \lambda) = \frac{1 - \|\mathbf{p}\|^2}{\|\mathbf{p} - \lambda\|^2} \tag{8}$$

This optimal value of K depends on the unknown value of $\mathbf{p}$, which can be estimated by $\hat{\mathbf{p}} = \mathbf{X}/N$. In terms of $\mathbf{x}$, the observed value of the random variable $\mathbf{X}$, $\hat{K}$ is

$$\hat{K} = \frac{N^2 - \sum_{i=1}^{t} x_i^2}{\sum_{i=1}^{t} x_i^2 - 2N\sum_{i=1}^{t} x_i\lambda_i + N^2\sum_{i=1}^{t} \lambda_i^2} \tag{9}$$

A pseudo-Bayes estimator of $\mathbf{p}$ is then

$$p* = \hat{\mathbf{q}}(\hat{K}, \lambda) = (\frac{N}{N + \hat{K}})\hat{\mathbf{p}} + \frac{\hat{K}}{N + \hat{K}}\lambda \tag{10}$$

where $\hat{K}$ is given in (9). Other pseudo-Bayes estimators of $\mathbf{p}$ are possible, and they correspond to alternative ways of estimating the optimal value of $\mathbf{K}$.

Given a training data that is composed of both normal and known attack instances, we can construct a contingency table, in which each column refers to an attribute characterizing an aspect of the instances and each row refers to a class of the training data that is either normal or a attack name. The non-categorical attributes are converted to the categorical ones. Besides all the possible classes of training data, an extra class will be added to represent new attacks. The table is built in such way that the cell value of ith row and jth column denotes the number of instances in training data that class i and attribute j both are present. The cell values of new attacks will be initialized with zeros. By pseudo-Bayes, the contingency table will be smoothed and each cell will be given an estimated value. The estimated cell values of unknown attacks will be $K\lambda_i/(N + K)$ by (4). Figure 5 shows the algorithm of pseudo-Bayes estimators.

Based on the "smoothed" contingency table by pseudo-Bayes estimators, we compute the prior and post probability of normal activity, known attacks, and novel attacks, and then build a Naive Bayes classifier. The classifier is able to detect novel attacks.

5 Performance of ADAM

ADAM is very effective in detecting *Denial of Service*(DOS), *Distributed Denial of Service* (DDOS) and PROBE attacks. In this section, we discuss the system performance of ADAM based on

Procedure of pseudo-Bayes estimator

begin

select λ_{ij} as following:

$\lambda_{ij} = \frac{1}{g_0}\frac{x_{ij}}{S_j}, \qquad S_j = \sum_{i=0}^{I} x_{ij}$

compute the weighted factor $\widehat{K} = (N^2 - \sum x_{ij}^2)/\sum_{i,j}(x_{ij} - N\lambda_{ij})^2$

compute the cell estimates $m_{ij}^* = Np_{ij}^* = N(x_{ij} + \widehat{K}\lambda_{ij})/(N + \widehat{K})$

end

Figure 5: Algorithm *Pseudo-Bayes estimator algorithm*

1999 DARPA Intrusion Detection System Evaluation results, though many new features (including pseudo-Bayes estimators, temporal association rules, etc.) have been added to the ADAM after 1999 DARPA evaluation,

5.1 Performance with DARPA Intrusion Detection Evaluation data

DARPA Intrusion Detection Systems (IDS) Evaluation project is the first effort to provide data and methodology for off-line evaluation of intrusion detection systems. It was led by the Information System Technology Group of MIT Lincoln Laboratory, under DARPA ITO and Air Force Research Laboratory sponsorship. Details information can be found in [10]. The project started from 1998 and continued in 1999. Today DARPA data sets are widely used as the benchmark for IDS evaluation. 1998 DARPA evaluation data contain seven weeks of training data and two weeks of test data. 1999 DARPA evaluation data contain three weeks of training data and two weeks of test data. Attacks in all training data are labeled. The test datasets of 1999 Evaluation contain two parts collected at a gateway inside the mimic subnet and at one outside the subnet respectively. Both training and test data are provided in several forms: UNIX BSM, tcpdump, and NT Data.

Attacks in DARPA data are classified into four types: *user-to-root attacks* (ROOT), *remote-to-local attacks* (LOC), *surveillance* or *probe attacks* (PROBE), and DOS.

Due to the attacks nature, ADAM is more efficient in PROBE and DOS attacks. The reason is as follows. PROBE and DOS attacks are mostly IP level attacks, which means we can detect them by checking TCP/IP headers (on which ADAM is based). *Password guess* (*PSSWD*) and *dictionary attacks* (*DIC*) attacks are application level attacks, but they happen to generate more than normal activities during a short time period, so they can be captured as generating "hot" and abnormal association rules. Of course, in order to identify them as *PSSWD, DIC* or *snmpget*[2], we have to resort to the TCP raw data (packets) to see what operations these connections performed. (E.g., long login trails in the *PSSWD* attack, each with a different password as input.) These operations

[2]DARPA describes snmpget as an attack in which the intruder monitors a router after guessing the SNMP password. If the SNMP configuration is allowed, an attacker with this password can modify the routing tables.

max. number of false positives allowed	1/day	10/day	100/day
total number of attacks detected	24	24	24
percentage of the total number of attacks detected	100%	100%	100%

(a) PROBE attacks detected versus the maximum number of false positives

max. number of false positives allowed	1/day	10/day	100/day
total number of DOS attacks detected	33	35	35
percentage of the total number of attacks detected	82.5%	87.5%	87.5%

(b) DOS attacks detected versus the maximum number of false positives

Figure 6: Experimental results on DARPA 1998 data

max. number of false positives allowed	1/day	10/day	100/day
total number of attacks detected	12	12	12
percentage of the total number of attacks detected	32.4%	32.4%	32.4%

(a) PROBE attacks detected versus the maximum number of false positives

max. number of false positives allowed	1/day	10/day	100/day
total number of DOS attacks detected	30	30	30
percentage of the total number of attacks detected	46.2%	46.2%	46.2%

(b) DOS attacks detected versus the maximum number of false positives

Figure 7: Experimental results on DARPA 1999 data

that act as a signature to *PSSWD* and *DIC* can be captured by the classifier. On the other hand, most of the other LOC and ROOT attacks occur at the application level and they can be finished in a single connection. That makes it impossible for them to be captured by using association rules (the support of the generated rule is not enough to pass the threshold and if we decreased the threshold to that level, we would generate lots of false positives).

Figure 6 give the ADAM's performance on 1998 DARPA data regarding to DOS and PROBE attacks. Figure 6(a) shows the results for TCP-dump Probe attacks, and Figure 6(b) shows them for the TCP-dump DOS attacks. 1/day means only 1 false alarm per day is allowed in a system, 10/day means 10 false alarms per day are allowed, and so on. These are exact criteria that DARPA used to evaluate an IDS performance, and they are used to compare the number of detected attacks by IDSs given the restricted number of false alarms. In this way, an IDS needs to sort the output alarms by their severity levels. The most serious alarm should be on the top of an alarm list.

As it can be seen in the tables, all the Probe attacks are detected regardless of the level of false positives tolerated. For the DOS attacks, the range of attacks goes from 70% to 75%, a number that makes ADAM rank very competitively with respect to the results obtained by the participants of the contest.

Figure 7 shows the results of DARPA 1999 test data. ADAM participated in DARPA 1999

Index	1	2	3	4	5	6	7	8	9	10
Name	smurf	ipsweep	portsweep	pod	mailbomb	teardrop	snmpget	back	neptune	process

Figure 8: Attacks that are used in progressive training

Name	#ins.	Round Number										
		0	1	2	3	4	5	6	7	8	9	10
pod	73	73	73	73	73	73	73	73	73	73	73	73
mailbomb	2	2	2	2	2	2	2	2	2	2	2	2
satan	10	2	2	2	2	2	2	2	2	2	2	2
apache2	9	0	0	0	0	0	0	0	0	0	0	0
teardrop	11	11	11	11	11	11	11	11	11	11	11	11
snmpget	13	1	1	1	1	1	1	1	1	1	1	0
snmpguess	2	2	2	2	2	2	2	2	2	2	2	2
neptune	23	7	7	7	7	7	7	7	7	7	9	9
process	6	6	6	6	6	6	6	6	6	6	6	6
portsweep	5	5	5	5	5	5	5	5	5	5	5	5
back	8	0	0	0	0	0	0	0	0	0	0	0
mscan	6	6	6	6	6	6	6	6	6	6	6	6
smurf	156	156	156	156	156	156	156	156	156	156	156	156
nmap	6	4	4	4	4	4	4	4	4	4	4	4
ipsweep	3	3	3	3	3	3	3	3	3	3	3	3
false alarms/day		4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7	4.7

Figure 9: Experiments on 1998 DARPA test data

intrusion detection evaluation, focusing on detecting DOS and PROBE attacks from tcpdump data and performed quite well.

5.2 Performance of ADAM on novel attacks

In this section, we report ADAM's performance when facing new attacks based on DARPA data. We configure the experiments in several different ways, and all experiment give very similar results and demonstrate that, with the help of pseudo-Bayes estimator, ADAM is able to detect novel attacks with low false alarms. For brevity, we only present the result of one experiment. Details of other experiment results can be referred to [6, 43].

The experiment is done in two stages. First, we derive pseudo-Bayes estimator and a Naive Bayes classifier from an attack free training data which is constructed by removing all the attack instances from 1998 DARPA training data. The reason we use 1998 training data is that it contains more weeks of data and is claimed to be more complete. The second stage is composed of several rounds. In each round, the pseudo-Bayes estimator and a Naive Bayes classifier are obtained by adding one attack instances to the previous round's training data, which starts with the attack free training data of stage 1 in the beginning and is accumulated by one attack instance in each round. We refer to the second stage as the progressive training step. Then we test the Naive Bayes classifiers of two stages against 1998 and 1999 DARPA test data.

Figure 8 shows the attacks that are added into the training data in the progressive training. *Index* gives the order of attacks that are added into the training data, and it also equals to the number of attack types that are included in the training data. Note that the order of attacks is chosen randomly. Figures 9 shows the experimental results of the classifiers of two stages on 1998

DARPA test data. The first two columns of the table give the attack name as well as the number of occurrence in the test data. The rest of the columns list the number of instances of each attack that has been captured in a round. *Round Index* refers to the round number of the progressive training. n means that it is the nth round and training data contains the attacks whose index number(shown in figure 8) is less than or equal to n. For instance, round number 2 means that both *smurf* and *ipsweep* are in the training data. Here we use *round 0* to represent the classifier of stage 1 since the training data does not contain any attacks. *false alarms/day* gives the average number of false alarms in each day.

It is interesting to see that the classifiers behave very similarly regardless of the number of different attacks are included in their training data. As each round's training data share the same normal instances which are supposed to be complete, it reveals a very important and desired property of pseudo-Bayes approach: the detection mainly depends on the normal instances in training data, and whether the training data contain attacks or not does not affect detection much if the normal instances give a complete description of the normal activities. Totally, around 80% percent attacks are either fully or partially detected, while at the same time the false alarm rates are pretty low. Besides, the figures show that more instances of the attack *neptune* are captured after round 9. It is understandable because as more information about attacks is present, the classifiers become more robust.

As we study the missed attacks of each test data, we find that they are very similar to the normal instances regarding to the attribute values. The experiments show that pseudo-Bayes estimator performs very well in deriving the characteristics of abnormal activities from the knowledge of normal ones. Pseudo-Bayes estimator method is good at capturing those attacks that can be distinguished from normal instances in terms of attribute values, but it does not perform well on the attacks that are similar to normal instances. Indeed it is hard for any classifier to distinguish attacks from normal instances if they are similar. The significance of pseudo-Bayes estimators lies in that it can derive the posterior probabilities of abnormal activity without any prior knowledge about them. If we can characterize the normal activity accurately, then this method can be very promising in detecting the abnormal activity that deviate from the normal ones. If the abnormal activity are similar to the normal ones, it would be difficult to distinguish them not only for pseudo-Bayes estimator, but for all classifiers. Most importantly, pseudo-Bayes estimators methods eliminates the dependency of ADAM on training data with labeled attacks.

6 Conclusions

We have presented ADAM system which is built on a data mining core to accomplish network anomaly detection. ADAM has two distinct properties: it uses temporal association rules to characterize time factors in profile, and it employs pseudo-Bayes estimator technique to capture novel attacks with low false alarms.

We are continuing this research on ADAM in three different ways. First, instead of selecting the threshold and window size in the test phase arbitrarily, we want to automate the selection

with the guidance of pro le so that different values of these parameters will be chosen according to the property of data, such as the time of connections, the direction of connections which can be inbound (from outside hosts to inside hosts), outbound (from inside hosts to outside hosts) and internal(from inside hosts to insides hosts), etc. Second, we will investigate new techniques that can completely eliminate the dependency of ADAM on attack free training data. Like any other anomaly detection systems, ADAM requires an attack free training data to learn the pro le, which is not easy to obtain in reality. Third, we want to enhance the system by enlarging the scope of attack types that can be detected.

References

[1] T. Abraham. IDDM: Intrusion detection using data mining techniques. Technical Report DSTO-GD-0286, DSTO Electronics and Surveillance Research Lab, Fort Washington, PA, 2001.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Washington D.C., may 1993.

[3] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, and A. Valdes. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection exppert system (nides). Technical Report SRI-CSL-95-06, SRI International, Menlo Park, CA, May 1995.

[4] J. Bala, S. Baik, A. Jadjarian, B.K. Gogia, and C. Manthorne. Application of a distributed data mining approach to network intrusion detection. In *1st International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1419–1420, Bologna, Italy, July 2002.

[5] D. Barbará, J. Couto, S. Jajodia, and N. Wu. Adam: Detecting intrusions by data mining. To appear.

[6] D. Barbará N. Wu, and S. Jajodia. Detecting novel network intrusions using bayes estimators. In *1st SIAM International Conference on Data Mining*, Chicago, IL, April 2001.

[7] David S. Bauer and Michael E. Koblentz. Nidx–an expert system for real-time network intrusion detection. In *IEEE Computer Networking Symposium*, pages 98–106, 1998.

[8] Y. M.M. Bishop and S. E. Fienberg. *Discrete Multivariate Analysis*: *Theory and Practice*. The MIT Press, 1975.

[9] M. Crosbie, B. Dole, T. Ellis, I. Krsul, and E. Spafford. Idiot - users guide. Technical Report TR-96-050, COAST Laboratory, Purdue University, September 1996.

[10] D. E. Denning and P. Neuman. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, pages 222–232, February 1987.

[11] D. E. Denning and P. G. Neumann. Requirements and model for ides – a real-time intrusion detection system. Technical report, SRI International, August 1985.

[12] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classi er under zero-one loss. *Machine Learning*, 29(2/3):103–130, 1997.

[13] William DuMouchel. Computer intrusion detection based on bayes factors for comparing command transition probabilities. Technical report, National Institue of Statistical Science, February 1999.

[14] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Proc. of IEEE symp. Security and Privacy*, 1996.

[15] J. Gama. Local cascade generalization. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.

[16] Joao Gama. Combining classi ers by constructive induction. In *Machine Learning ECML98*. Springer Verlag, 1998.

[17] A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proc.. USENIX Security Symp.*, 1999.

[18] J. Han and Y. Fu. Discovery of Multiple-Level Association Rules from Large Databases. In *Proceedings of the 21st Very Large Data Bases Conference, Zurich, Swizerland*, 1995.

[19] G. Helmer, J. S.K. Wong, V. Honavar, and L. Miller. Automated discovery of concise predictive rules for intrusion detection. *The Journal of Systems and Software*, 60:165–175, 2002.

[20] H. S. Javitz and A. Valdes. The sri ides statistical anomaly detector. In *IEEE Symposium on Security and Privacy*, pages 316–326, Oakland, CA, May 1991.

[21] George H. John and Pat Langley. Estimating continuous distributions in bayesian classi ers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo, 1995. Morgan Kaufmann.

[22] K Julisch. Mining alarms clusters to improve alarm handling efficiency. In *17th Annual Computer Security Applications Conference*, New Orleans, LA, Dec 2001.

[23] K Julisch and M. Dacier. Mining intrusion detection alarms for actionable knowledge. In *Eighth International Conference on Knowledge Discovery and Data Mining (KDD'02)*,Edmonton, Alberta, July 2002.

[24] M. Klemettinen. A *Knowledge Discovery Methodlogy for Telecommunication Network Alarm Data*. PhD thesis, University of Helsinky, 1999.

[25] T. Lane and C. Brodely. Approaches to online learning and concept drift for user identi cation in computer security. In *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, New York, NY, August 1998.

[26] W. Lee, S.J.Stolfo, and K.W. Mok. Mining audit data to build intrusion detection models. In *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, New York, NY, August 1998.

[27] W. Lee, S. Stolfo, and K. Mok. A Data Mining Framework for Adaptive Intrusion Detection. *Artificial Intelligence Review*, 1999.

[28] W. Lee, S. Stolfo, and K. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1999.

[29] Wenke Lee and S.J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of Seventh USENIX Security Symposium*, San Antonio, TX, January 1998.

[30] Y. Li, N. Wu, S. Jajodia, and X. S. Wang. Enhancing pro les for anomaly detecting using time granularities. Accepted by Journal of Computer Security.

[31] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P. G. Neumann, H. S. Javitz, A. Valdes, and T. D. Garvey. A real time intrusion detection expert system (ides). Technical report, SRI International, February 1992.

[32] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion detection expert system. pages 59–66, New York, April 1988.

[33] T.F. Lunt.Real-Time Intrusion Detection. In *Compcon '89*: *IEEE Computer Conference*, pages 348–353, February 1989.

[34] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz. A data mining analysis of rtid alarms. *Computer Networks*, 34:571–577, 2000.

[35] P. Neumann and P. Porras. Experience with Emerald to Date. In *First USENIX Workshop on Intrusion Detection and Network Monitoring*, *Santa Clara*, *California*, April 1999.

[36] P. Porras and R. Kemmerer. Penetration state transition analysis -a rule-based intrusion detection approach. In *8th Annual Computer Security Applications Conference*, pages 220– 229, November 1992.

[37] P. Porras and P. G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *the 19th National Information Systems Security Conference*, pages 353–365, Baltimore, MD, October 1997.

[38] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Speci cation-based anomaly detection: a new approach for detecting network intrusions. In *9th ACM Conference on Computer and Communications Security*, pages 265–274, Washington, DC, November 2002.

[39] Matthew G. Shultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy*, 2001.

[40] Alfonso Valdes and Keith Skinner. Adaptive, model-based monitoring for cyber attack detection. In *3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000)*, pages 83–90, October 2000.

[41] G. Vigna and R. Kemmerer. NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the 14th Annual Computer Security Application Conference*, *Scottsdale*, *Arizona*, December 1998.

[42] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000)*, pages 93–110, October 2000.

[43] Ningning Wu. *Audit Data Analysis and Mining*. PhD thesis, George Mason University, 2001.

Part II. Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances

1. Introduction

Since intrusion detection systems generally focus on low-level events and report them independently, network administrators are often overwhelmed by large volumes of alerts. This has motivated recent work in alarm aggregation, to reduce administrator workload and provide higher-level situational awareness. Ideally, alarm aggregates should help one distinguish coordinated, multi-step attacks from isolated events. It is also critical to know if one's network is actually vulnerable to detected attacks, and not just from the standpoint of individual machines but also in the context of the overall network and its most critical resources.

Various approaches have been proposed to correlate intrusion alarms and build attack scenarios from them. For building attack scenarios, a particularly effective form of correlation is causal correlation, which is based on analyzing dependencies among intrusion events.

One approach to causal event correlation is to apply logical rules that chain together events based on their relevant attributes. But there are several problems with rule-based approaches to event correlation. It can be difficult for complex rule systems to keep pace with online streams of events, and maintaining the rule sets needed for constructing attack scenarios from disparate events can be difficult. Also, missing events can prevent rules from assembling a proper attack scenario, and attempts at inferring hypothetical missing attacks can lead to irrelevant results.

Another approach to causal correlation is to represent relationships among events with graphs instead of logical rules. However, because this is still based on intrusion detection information only, it can potentially give irrelevant results when hypothesizing missing events. Also, because the attack scenario graphs are constructed as events occur, it may be difficult for to keep pace with online event streams.

In existing approaches, the implicit assumption is that intrusion events are caused by the execution of attacker exploits. These approaches then model intrusion events in terms of rules (preconditions/postconditions) for the implicit exploits. But the fundamental problem with these approaches is they do not include network vulnerabilities in their model, which would provide the proper context for their implied exploits. This is the source of potentially irrelevant scenarios or ambiguity for hypothesized missing events.

In this paper, we extend previous approaches to building attack scenarios by explicitly including network vulnerability/exploit relationships (i.e., the attack graph) in the model. In other words, the network attack graph is precisely the model component that adds the necessary context to the exploits implied by intrusion events. A crucial design criterion is to maintain low overhead for online event processing. Our online processing depends solely on a manageable set of pre-computed attack graph distances. To process an online intrusion event, only a distance lookup and a small number of arithmetic operations are required.

We first build a joint model of attacker exploits and network vulnerabilities. The network vulnerability model is created either manually or automatically from the output of the Nessus vulnerability scanner. From the joint exploit/vulnerability model, we then compute distances (number of steps in the shortest path) between each pair of exploits in the attack graph (for all possible network

attacks). These distances provide a concise measure of exploit relatedness, which we use for subsequent online causal correlation of intrusion detection events.

As detection events occur, we map them to attack graph exploits, and look up the distances between pairs of corresponding exploits. This allows us to correlate events through attack graph information, without the online overhead of rule execution or graph building. We iteratively build event paths, with a numeric correlation score for each event. Missing events are handled in a natural way, i.e., we quantify gaps in attack scenarios through attack graph distances. Events that cannot be mapped to the attack graph initially can be considered in post-analysis and possibly merged with existing attack scenarios.

Sequences of correlation scores over event paths indicate likely attack scenarios. We apply a low-pass signal filter (the exponentially weighted moving average filter) to correlation sequences, which improves quality in the face of detection errors. We apply a threshold to filtered correlations to separate event paths into attack scenarios, i.e., only paths with sufficient correlation (sufficiently small attack graph gaps) are placed in the same attack scenario. We also compute an overall relevancy score for each resulting attack scenario, which measures the extent that it populates a path in the attack graph.

In the next section, we review related work in this area. Section 3 then describes our underlying model, and Section 4 gives details of our implementation of this model. In Section 5, we provide experimental evidence in support of our approach, and in Section 6 we summarize this work and draw conclusions.

2. Related Work

Our approach extends recent work in causal correlation of intrusion events. But rather than correlating based on dependencies among events only, we take the novel direction of including the interdependent network vulnerabilities (i.e., network attack graph) in the correlation model.

In [1], the approach to causal correlation is to define logical rules that relate generic (network independent) events through preconditions/postconditions. As events occur, the generic rules are instantiated with attributes such as time, source/destination machine, and vulnerability type, and evaluated via Prolog to chain events together. This approach does include additional implication rules for handling missed attacks. However, because it lacks knowledge of the network vulnerabilities, it is unable to narrow down hypothesized attacks to ones that are truly relevant. Also, while this approach generates rules offline (from a set of generic exploit specifications), in online mode it still needs to evaluate the rules. The approach in [1] does include merging of identical events, which is complementary to our approach. The event merging is accomplished through clustering correlation, a form of correlation that has been described by other authors, e.g., [2][3][4].

The approach in [5] is to represent relationships among events as a graph rather than through rules. Such graphs are less complex than rule systems, and indeed we apply a similar graph representation in our approach. But the approach in [5] does not correlate events with vulnerability information, as we do. It can therefore give irrelevant results when hypothesizing missing events, because events are not grounded in real network vulnerabilities. Also, the attack scenario graphs are constructed as events occur, making it more difficult to keep pace with online event streams. In contrast, we capture relationships among attack graph elements in concise distance measurements, so that no graph manipulation is done online.

Work has been done in integrating intrusion detection with vulnerabilities information, notably [6]. However, this work considers vulnerabilities in isolation, without considering the overall impact of

combined vulnerabilities on a network. Also, it does not address the critical problem of building attack scenarios from individual events. There are actually 2 vendors (Tenable Network Security and Internet Security Systems) that integrate their respective intrusion detection and vulnerability scanning tools, but again this considers vulnerabilities only in isolation.

On a related research front, work has been done in automatic construction of attack graphs from network vulnerability models. Our attack graph construction is based on such prior work [7][8][9]. Other approaches to attack graph construction have been proposed, including logic-based [10][11] and graph-based [12][13][14] approaches. These have been generally effective for assessing overall network security posture or hardening networks, although not all the proposed approaches are scalable. Our attack graph representation is based on exploit dependencies rather than security state enumeration, so that we avoid combinatorial explosion. The basic representation was first described in [14], and later modified in [8][15].

3. Underlying Model

Construction of network attack graphs is based on the application of attacker exploit rules. These rules map the conditions for exploit success (preconditions) to conditions induced by the exploit (postconditions). For example, an exploit may require user privilege on the attacker machine and yield root privilege on the victim machine. An attack graph is constructed by finding the interdependencies of exploits with respect to machines on a network.

While we employ a scalable (low-order polynomial) attack graph representation, the cost of attack graph computation still prohibits online calculation per intrusion event. The attack graph needs to be fully realized before events occur. Once an alarm is raised, its event is mapped to an exploit in the attack graph. Multiple precondition/postcondition dependencies between exploits are represented with a single graph edge, meaning that the "to" exploit depends on at least one postcondition of the "from" exploit.

A typical scenario for network vulnerability analysis includes an initial attacking machine (either outside or inside the administered network) and a set of attack goal conditions (e.g., root) on one or more machines. Given that an exploit's preconditions are met, the state of the victim machine changes per the exploit's postconditions. Upon success of an exploit, the conditions of the victim machine may meet other exploits launched from that machine. Successful exploits launched from the victim machine are linked to the exploits that provide its preconditions. By executing and linking exploits in this fashion, an attack graph is formed.

For constructing attack scenarios, we do not base the attack graph on a fixed attacker/goal scenario as is typically done in network vulnerability analysis. Neither the goal nor the attacker is known when the attack graph is computed, before intrusion events are actually considered. The assumption is that attacks can come from any machine inside or outside an administered network. The attacker may have infiltrated the network through stealth attacks, or the attack may have come from an insider who abuses his granted privileges. Similarly, the attack goal is open, since it could be any adverse condition (such as denial of service, root privilege, or unauthorized data access) on any machine. In short, our model considers the full scope of possible attack paths.

Two events that fall on a connected path in an attack graph are considered correlated (at least to some extent). Clearly, events should be fully correlated if they map to adjacent exploits in the attack graph, since this is the strongest relationship possible. Conversely, events mapped to non-adjacent exploits are only partially correlated, as shown in Figure 1. In this case, we determine the degree of event correlation through graph distance between corresponding exploits.

The graph distance between a pair of exploits is the minimum length of the paths connecting them. If no such path exists, then the distance is infinite. Graph distance measures the most direct path an attacker can take between two exploits. While longer paths might be possible between exploits, the shortest path is the best assumption for event correlation, and is the most efficient to compute. The use of minimum path length does not hinder the ability to analyze longer paths, since these paths are constructed by assembling shorter paths. Using minimum path length also resolves cycles in the attack graph, which would otherwise indicate redundant attack steps. Our graph distances are unweighted, i.e., no weights are applied to graph edges between exploits.

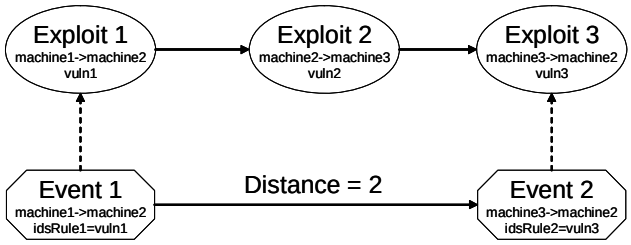

Figure 1: Partially correlated events.

The exploit distances are pre-computed once for an attack graph, and then applied continuously for a real-time stream of intrusion events. The exploit distances supply the necessary information to form event paths. An event is added to the end of a path if it maps to an exploit that has a finite distance from the exploit mapped to the last event in the path. Event time is naturally accounted for, because events are added at the ends of paths, which were constructed from prior events. If a new event is unreachable from all existing event paths (i.e., if the corresponding attack graph distances are infinite), then the event forms the beginning of a new path.

In Figure 2, suppose an initial event path exists as Event 1, corresponding to Exploit 1. A new Event 2 arrives, corresponding to Exploit 3. Since Exploit 3 is reachable from Event 1 with a graph distance of 2, Event 2 is added to the event path. A new event may trigger the creation of additional independent event paths. Continuing with our example, suppose a new Event 3 arrives, which corresponds to Exploit 4. Exploit 4 is reachable from both Exploit 1 and Exploit 3. Therefore, Event 3 can be correlated to Event 1 independently of Event 2. Since Event 2 might have nothing to do with Event 1, a new path is created as a record of another potential attack scenario. Thus we have the 2 paths Event $1 \rightarrow$ Event $2 \rightarrow$ Event 3 and Event $1 \rightarrow$ Event 3. In the figure, these 2 paths are drawn with solid lines and dashed lines, respectively, in the event graph.

In our model, cycles in the event graph are unrolled. For example, in Figure 2, Exploit 4 can reach back to Exploit 1 through a distance of 3. Event 4 occurs after Event 3, and is identical to Event 1, i.e., it also maps to Exploit 1. For example, Exploit 4 might yield new privileges based on trust gained from the intervening 3 exploits. Thus two new paths are formed:

1. Event $1 \rightarrow$ Event $2 \rightarrow$ Event $3 \rightarrow$ Event 4 (solid lines)
2. Event $1 \rightarrow$ Event $3 \rightarrow$ Event 4 (dashed lines)

These are shown with solid and dashed lines, respectively, in Figure 2.

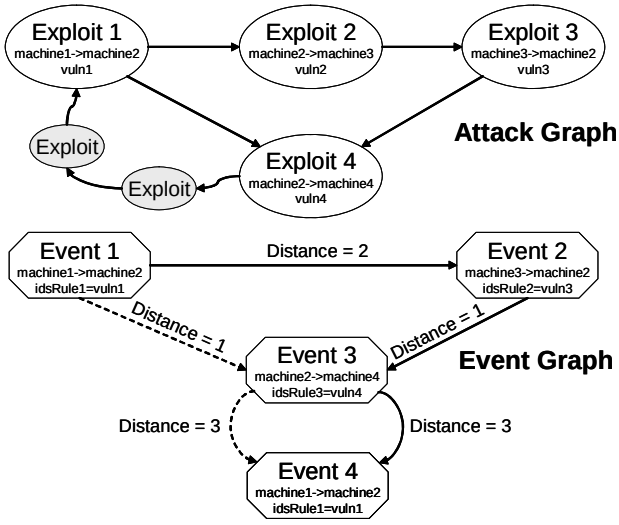

Figure 2: Creating event paths.

For the example in Figure 2, the events correspond to exploits that lie within relatively close distances to each other. But this may often not be the case. Indeed, it is reasonable to assign events whose exploits are widely separated in the attack graph to separate attack scenarios. Since event distances greater than unity represent missed detection events (according to the attack graph), it is possible that such distances sometimes occur within a set of coordinated attacks, since real attacks are sometimes missed. But when event distances become larger, larger numbers of attacks would need to be missed if they were really coming from a coordinated attack.

Thus, we apply a correlation threshold that segments event paths into highly correlated attack scenarios. In other words, a consecutive sequence of events that lies above the threshold defines an attack scenario. When individual event paths are formed from the incoming stream of events, new event paths are created when a new event is not reachable (infinite distance) from the currently existing set of event paths. In this way, event paths have an obvious beginning based on (non-) reachability. The correlation threshold provides a way to end an event path when the distance to the next event is too large, but is still finite.

The distances between events in an event path are crucial information. But because of possible false detections (positive and negative), the individual distance values are somewhat suspect. We could gain more confidence in our estimate by averaging the individual distance values. While this would capture the global trend of the event path, local trends would be lost. Also, it is convenient to invert the event distances (use their reciprocals), so that they lie in the range [0,1], with larger values representing stronger correlation. Thus the inverse distances represent similarities rather than dissimilarities.

But rather than computing the global average of inverse event distances, we compute a moving average, which has the ability to capture local trends while still providing error resiliency. An unweighted moving average defines a data window, and treats each data point in the window equally when calculating the average. However, it is reasonable to assume the most current events tend to better reflect the current security state. We therefore apply the exponentially weighted moving average, which places more emphasis on more recent events by discounting older events in an exponential manner. It is known to be identical to the discrete first-order low-pass signal filter.

Let d_k be the attack graph distance between a pair of intrusion events. Then the inverse event distance is $x_k = 1/d_k$. We then apply the exponentially weighted moving average filter to a sequence of these x_k:

$$\bar{x}_k = \alpha \bar{x}_{k-1} + (1-\alpha)x_k . \qquad (1)$$

The sequence of values of $\bar{x}_k$ is the filtered version of the original sequence of inverse event distances x_k, for some filter constant $0 \le \alpha \le 1$. The filtered inverse event distances $\bar{x}_k$ are the basic measure of event correlation in our model. For convenience, we define a correlation of unity for the first event in a path (i.e., it is fully correlated with itself), even though there is no previous event to compare it to.

The inverse intrusion event distances are filtered very efficiently through the recursive formulation in Equation (1). Computation requires no storage of past values of *x*, and only one addition and 2 multiplications per data point are required.

In the exponentially weighted moving average filter, the filter constant $0 \le \alpha \le 1$ dictates the degree of filtering. As $\alpha \to 1$, the degree of filtering is so great that individual event (inverse) distances do not even contribute to the calculation of the average. On the other extreme, as $\alpha \to 0$, virtually no filtering is performed, so that $\bar{x}_k \to x_k$. Values in the range of $0.3 \le \alpha \le 0.4$ generally work well in practice.

The filtered inverse distances in Equation (1) provide a good local measure of event correlation. In particular, they perform well for the application of the score threshold for segmenting event paths into attack scenarios. But once an attack scenario is formed, the individual filtered inverse distances do not provide an overall measure of correlation for it. We introduce another score that provides a measure of relevancy for the entire scenario, based on attack path occupancy by events.

For attack scenario s_k, $|s_k|$ is the number of events in the scenario. Next, let l_k be the cumulative distance between pairs of events in the scenario. Then the attack scenario relevancy score r_k is

$$r_k = |s_k| / l_k . \qquad (2)$$

Because the cumulative distance l_k is the length of the attack path that the scenario maps to, this relevance score r_k is the proportion of the attack path actually occupied by an attack scenario's intrusion events.

Our model is robust with respect to inconsistencies between events and vulnerabilities. Events that cannot be mapped to an exploit in the attack graph simply remain as isolated events. This might occur because there is no known mapping from a particular event to a vulnerability, or because a certain vulnerability was not known when constructing the attack graph. The converse is that there are certain vulnerabilities in the attack graph that have no corresponding intrusion detection signature. In this case, distances between events (in event paths) can be normalized by the expected distance between corresponding exploits in the attack graph.

4. Implementation Details

Figure 3 shows the system architecture for our implantation of the model described in the previous section. The *Attack Graph Analyzer* requires a joint model of the network and attacker exploits. *Exploit Modeling* is done through manual analysis of reported vulnerabilities and known exploits. We have researched almost 2000 Nessus vulnerabilities, from which we have modeled about 650 exploits (a significant portion of Nessus vulnerabilities are irrelevant for this kind of modeling). Because we

usually model exploits at a relatively high level of abstraction (e.g., in terms of access type, privilege level, and network connection), this manual process generally proceeds quickly.

Accurate modeling depends on sufficient information about vulnerabilities and exploits. Our exploit modeling is supported by an extensive database, which includes 37,000 vulnerabilities and 7,400 exploits, taken from 24 information sources including X-Force, Bugtraq, CVE, CERT, Nessus, and Snort. *Network Modeling* can be done manually, or generated automatically from Nessus vulnerability scanner output. In the case of network models created manually, we support model specification in terms of vulnerable software components (OS, patch level, web servers, configuration files, etc.), with rules to map these to Nessus vulnerabilities.

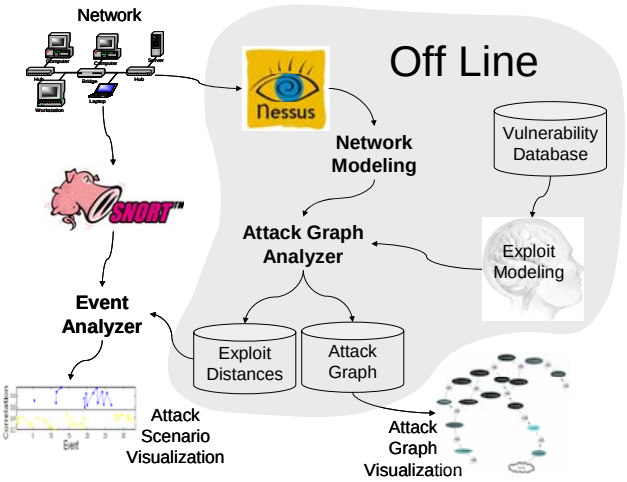

Figure 3: System architecture.

From the combined network and exploit models, we analyze attack paths and load the resulting exploit distances into an Oracle database. For efficiency, infinite distances (caused by some exploits not being reachable to others) are not recorded in the database. Rather, they are represented by their absence. In practice, a value can be chosen as an effective infinity, giving the distance computation algorithm a reasonable stopping point in declaring an exploit unreachable. Once exploit distances are calculated, they become a static image of the attack graph to be correlated with intrusion events. We can also store the attack graph itself for future offline attack graph visualization and post-analysis. All of this processing is done offline, as shown by the shaded region in Figure 3.

When Snort intrusion detection events are logged in the database, this triggers Oracle stored procedures in the *Event Analyzer* to process them. For each Snort event, we map the Snort identifier to the corresponding Nessus vulnerability identifier. In the case that a Snort identifier maps to multiple Nessus identifiers, we report all the identifiers, and conservatively select the shortest distance from among the candidate exploits for computing the correlation score. The lookup of pre-computed attack graph distances is based on source and destination IP addresses and Nessus vulnerability identifier. Note that only the distances between exploits are looked up, and no processing of the actual attack graph occurs online.

Event paths are formed in the manner described in the previous section, i.e., by adding new events to the ends of paths if the new event is reachable from the last event in the path, etc. For each path of intrusion events, the *Event Analyzer* inverts the distances between events (converts them from dissimilarities to similarities), then applies the exponentially weighted moving average filter in

Equation (1) to the inverse distances. The correlation threshold is then applied, as described in the previous section, which segments event paths into highly correlated attack scenarios. In practice, proper values of correlation threshold should be based on expected rates of missed detections.

5. Experiments

In this section, we demonstrate our approach through various experiments. The first experiment focuses on the application of correlation threshold for separating event paths into highly-correlated attack scenarios and the interaction between threshold value and low-pass filter constant. To instill a deeper understanding of this, we examine a small number of attacks in greater detail, as opposed to showing statistical results for large number of attacks. In the second experiment, we show more clearly how low-pass filtering makes it easier to distinguish regions of similar attack behavior in the presence of intrusion detection errors. The third experiment is a larger-scale scenario to demonstrate overall performance.

5.1 Scenario Building via Correlation Threshold

Figure 4 is a concise summary of the attack graph for this experiment. The network model in this experiment is generated from Nessus scans of real machines. In the figure, an oval between a pair of machines represents the set of exploits between that machine pair. In most cases, there are 2 numbers for exploit sets, reflecting the fact that some exploits are in one direction (from one machine to another), and other exploits are in the opposite direction. Unidirectional sets of exploits are drawn with directional arrowheads; for sets of exploits in both directions, arrowheads are omitted. This is a variation of the aggregated attack graph representation described in [15].

In this experiment, only remote-to-root exploits are included, to make results easier to interpret. That is, each exploit has preconditions of (1) execute access on the attacking machine and (2) a connection from the attacking machine to a vulnerable service on the victim machine, and postconditions of (1) execute access and (2) superuser privilege on the victim machine. Since connections to vulnerable services exist in the initial network conditions, and each exploit directly yields superuser access on the victim machine, the shortest exploit distance between machines is always one. In interpreting these distances from the figure, the actual numbers of exploits between pairs of machines are therefore irrelevant.

The important information from Figure 4 is the attack graph distances between the 8 intrusion events, which we can determine directly from the figure. The arrow beside "Event *x*" indicates the direction (source and destination machine) of the event. So the distance from Event 1 (an exploit from machine m23 to m80) to Event 2 (an exploit from machine m80 to m52) is one, the distance from Event 2 to Event 3 is 2, etc.

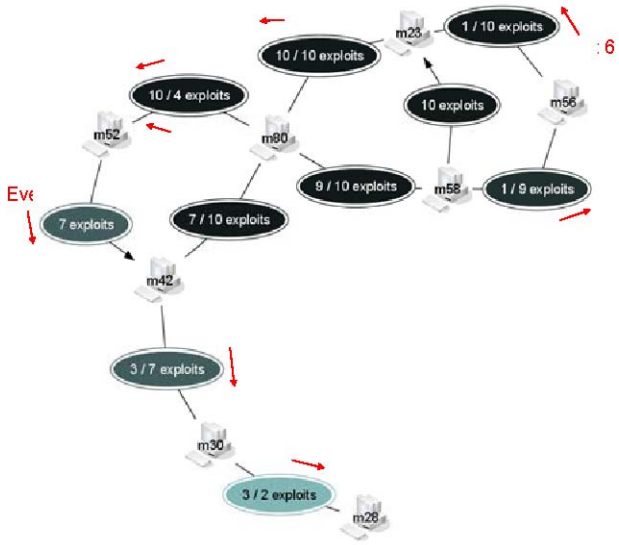

Figure 4: Aggregated attack graph.

Counting the distance from Event 4 to Event 5 is a bit more subtle. Here one must realize that "3/2 exploits" means there 3 exploits from m30 to m28, one of which is associated with Event 4. Then from the Event-4 exploit, in counting the shortest path to Event 5, there is one exploit from m28 to m30, one from m30 to m42, etc., for a total distance of 5. Figure 5 shows the full attack graph for this experiment, although it is cumbersome to use this complex graph for visually counting event distances.

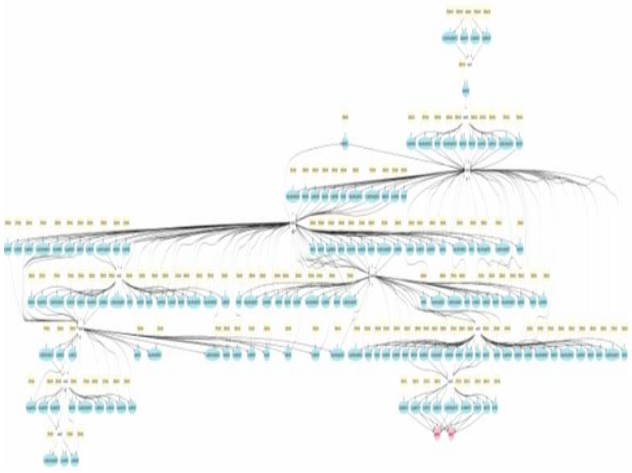

Figure 5: Non-aggregated attack graph.

Figure 6 shows the sequence of distances for the events in this experiment. Because every event is reachable from the previous event, only a single event path is generated.

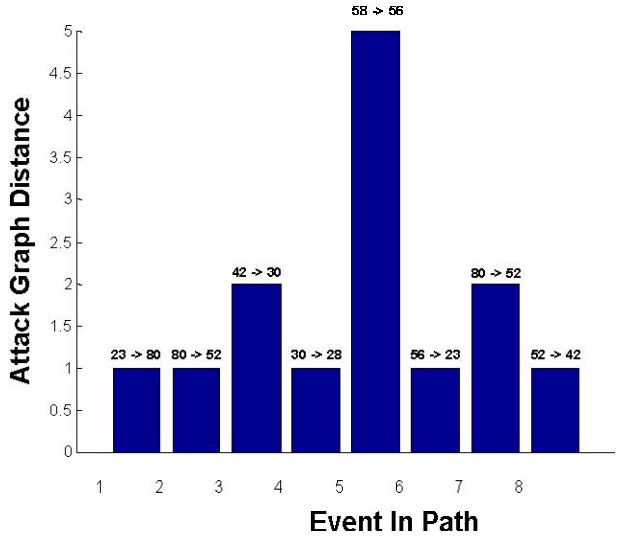

Figure 6: Attack graph distances for events.

Figure 7(a) shows the inverse of the attack graph distances from Figure 6, filtered via Equation (1), for different values of filter constant α . The vertical axis is the filtered inverse distance (i.e., the correlation score), the horizontal axis is the event number, and the axis into the page is $0.1 \leq \alpha \leq 0.9$. We apply a correlation threshold value of $T = 0.6$, shown as a horizontal plane.

For $\alpha = 0.1$ (front of page), very little filtering is applied, so that the filtered sequence looks very similar to the original sequence of inverse distances. In this region of α values, for the threshold $T = 0.6$, the event path is separated into 4 short attack scenarios:

1 Event 1 → Event 2
2 Event 4
3 Event 6
4 Event 8

The remaining events (3, 5, and 7) fall below the threshold and are considered isolated. However, the more likely scenario is that the distances=2 for Event 3 and Event 7 represent missed detections, since they are in the region of fully-correlated events. The distance=5 for Event 5 would require an unlikely high number of missed detections, so it is probably really is the start of a separate (multi-step) attack.

The problem is that, without adequate filtering, event distances are not being considered in the context of the recent history. One could lower the threshold to below $T = 0.5$ in this case, which would yield these most likely attack scenarios:

1 Event 1 → Event 2 → Event 3 → Event 4
2 Event 6 → Event 7 → Event 8

However, in general values below $T = 0.5$ are not particularly strong correlations, so this is not advisable.

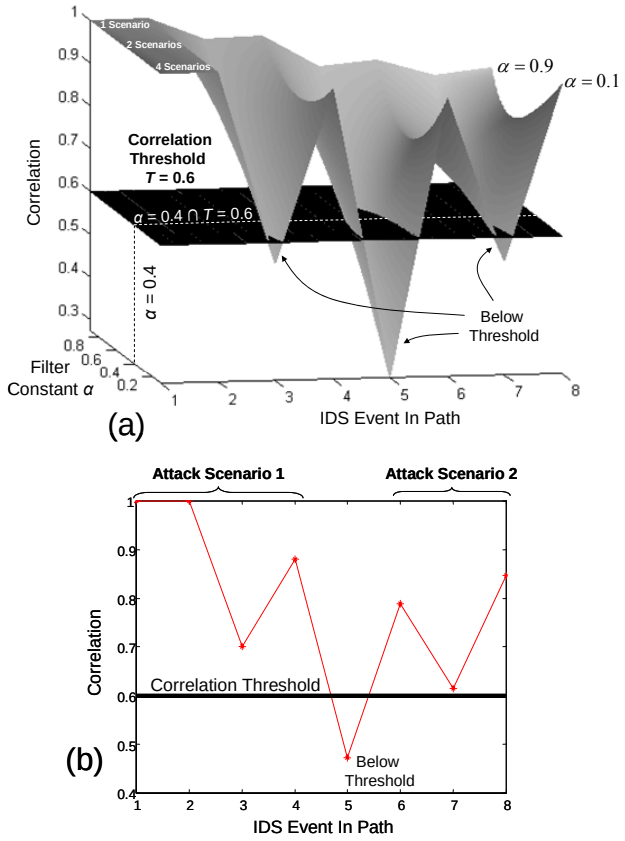

Figure 7: Distance filtering and threshold

For larger values of α (going into the page), more filtering is applied, so that distance recent history is considered more strongly. In this case, the threshold does separate the path into the 2 most likely attack scenarios. A cross section for $\alpha = 0.4$ is shown in Figure 7(b). For overly large values of α (e.g., in the region of $\alpha = 0.9$), so much filtering is applied that the entire path is considered a single attack scenario. In other words, it misses Event 5 as the start of a new attack scenario.

5.2 Signal Filtering for Detection Errors

Next, we describe an experiment that more clearly shows the need for low-pass signal filtering for handling intrusion detection errors. In particular, this experiment demonstrates how low-pass filtering makes it easier to distinguish regions of similar attack behavior through the application of a correlation score threshold.

The results of this experiment are shown in Figure 8. Here, the horizontal axis is the event in an event path. The vertical axes of the 4 plots are (respectively) raw attack graph distance between events, global average of inverse event distance, filtered inverse event distance, and unfiltered inverse event distance.

As a ground truth, the event path is divided into 7 regions. Region 1 (Events 1-7) is an uncoordinated series of events, i.e., one in which the events are unrelated and scattered across the network, so that distances between events are relatively long. Region 2 (Event 8) is a pair of events that occur immediately together in the attack graph (i.e., event distance=1, fully correlated). Region 3

(Events 9-14) is an uncoordinated series of events. Region 4 (Events 15-17) is a series of fully correlated events, and Region 5 (Events 18-24) is an uncoordinated series of events.

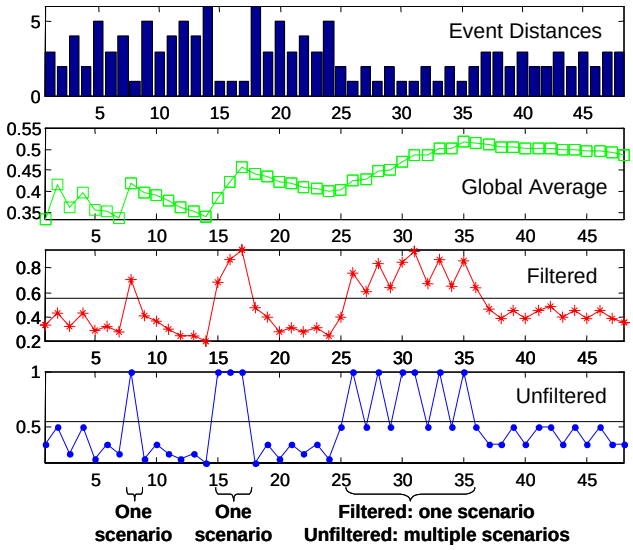

Figure 8: Filtering inverse event distances.

Regions 6 and 7 (Events 25-36 and Events 37-48, respectively) are a bit more subtle. In Region 6, the attack graph distances between events fluctuate between one and two. This represents a series of events for a single (multi-step) attack, or at least the work of a fairly consistently successful attacker. We could assume the distance=2 event pairs are from missed detections. In Region 6, the attack graph distances between events fluctuate between 2 and 3. In this case, it seems more likely to be an uncoordinated series of events that happen to occur more closely on the attack graph than say Region 1.

In Figure 8, we include global average (2nd from top in the figure) as a comparison to moving average. While each value captures the overall average inverse distance up to a given event, that does not allow us to make local decisions (e.g., through a correlation threshold) for separating the path into individual attack scenarios. Even the occurrence of fully-correlated Region 4 events cannot be distinguished through the application of a threshold.

For the unfiltered inverse distances (bottom of Figure 8), we can correctly distinguish the isolated pair of fully correlated events in Region 2, as well as the unbroken path of fully correlated events in Region 4. But there are problems for Region 6. This is the region in which fully correlated events are mixed with distance=2 events. This could be expected in a real sequence of attacks, when some of the attacks go undetected. Here, the unfiltered correlations fluctuate strongly, causing problems for setting a threshold for segmenting event paths into likely scenarios. At the threshold shown of 0.55, this region is segmented into multiple very small attack scenarios. The threshold could be lowered (to below 05), but that would cause problems for Region 7. Here, distance=2 and distance=3 event pairs are occurring. In this case, it is much less likely a coordinated attack is occurring. It would mean one or 2 attacks are repeatedly being missed, with no fully correlated events occurring. Lowering the threshold to handle Region 6 would cause Region 7 to be segmented into multiple very small scenarios.

In contrast, when the threshold is applied to the filtered version of the inverse event distances (2nd from bottom in Figure 8), this correctly forms attack scenarios corresponding to Regions 1 through 7. When filtering is applied, the distance for a new event takes into account the recent history of events, so that distances occurring after shorter distances tend to become shorter and distances occurring after longer distances tend to become longer. The degree of this effect is controlled by the filter constant α .

5.3 Performance

This experiment demonstrates overall performance for the implementation of our approach (see Section 4 for implementation details), using a large number of network attacks. In particular, we apply our implementation to a network of 9 victim machines, separated into 3 subnets, as shown in Figure 9.

In this experiment, subnet x.x.100.0 services internet traffic with a web server and an FTP server. Subnet x.x.128.0 supports administrative servers and an Oracle database server. Subnet x.x.200.0 is for administrative purposes. Traffic between subnets is filtered as shown in Figure 9. Traffic within each subnet is unfiltered, so that there is full connectivity to vulnerable services among machines in a subnet.

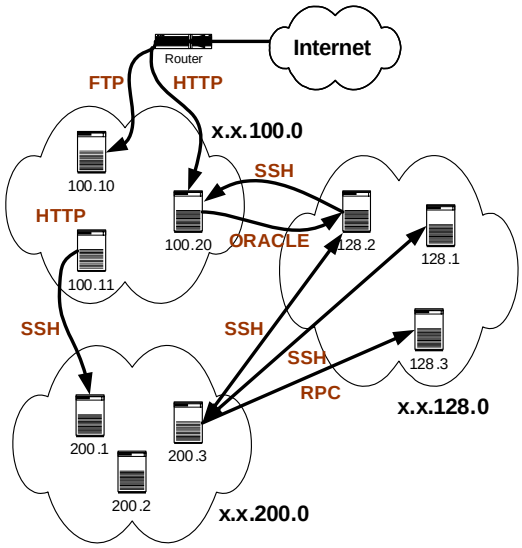

Figure 9: Network connectivity for third experiment.

The attack graph in this experiment contains 105 (machine-dependent) exploits. While there are 105^2=11025 possible distances between 105 exploits, the exploits leading from the internet are not reachable from the remaining exploits, and such infinite distances are not recorded (using an adjacency list representation). In particular, there are 10,395 recorded exploit distances.

We then injected 10,000 intrusion events, mixed with random traffic. We included isolated events as well as multi-step attacks. Using a filter constant of $\alpha = 0.4$ and a correlation threshold of 0.55, we correctly distinguished the multi-step attacks from the isolated events.

In online mode, it takes less than 4 minutes to process 10,000 events (about 24 milliseconds per event). This is on a machine with a 2-GHz processor, 1 megabyte of main memory, and two 100-gigabyte 15,000 RPM SCSI disk drives. Neither memory nor disk traffic showed more than 30% load.

6. Summary and Conclusions

In this paper, we extend previous approaches to attack scenario building by explicitly including the network attack graph in the model. The attack graph provides the necessary context for intrusion events, and provides the graph distances upon which our correlations are based. Our online event processing depends on pre-computed attack graph distances only, and requires only a lookup and 4 arithmetic operations.

To compute attack graph distances (offline), we build a model of attacker exploits and network vulnerabilities. We can create the network vulnerability model automatically from output of the Nessus vulnerability scanner. We then compute the distance of the shortest path between each pair of exploits in the attack graph. These distances are a concise measure of exploit relatedness, which we use for subsequent online causal correlation of intrusion detection events.

From the online stream of intrusion events, we build individual event paths based on attack graph reachability. The inverse distance between each event in a path is a measure of correlation. We apply a low-pass filter to sequences of inverse distances to provide resiliency against detection errors. The application of a threshold to the filtered distances separates event paths into highly correlated attack scenarios. We also compute an overall relevancy score for each resulting attack scenario.

We demonstrate our approach through several experiments. The results show that the approach generates attack scenarios with a high degree of causal correlation. We demonstrate the effectiveness of correlation thresholding, and well as its relationship to degree of applied filtering. We demonstrate real-time performance, processing an event every 24 milliseconds.

References

[1] F. Cuppens, A. Miege, "Alert Correlation in a Cooperative Intrusion Detection Framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[2] Y.-S. Wu, B. Foo, Y. Mei, S. Bagchi, "Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS," in *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2003.

[3] A. Valdes, K. Skinner, "Probabilistic Alert Correlation," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, Davis, California, 2001.

[4] O. Dain, R. Cunningham, "Building Scenarios from a Heterogenous Alert Stream," in *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, West Point, New York, June 2001.

[5] P. Ning, D. Xu, C. Healey, R. St. Amant, "Building Attack Scenarios through Integration of Complementary Alert Correlation Methods," in *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, February, 2004.

[6] B. Morin, L. Mé, H. Debar, M. Ducassé, "M2D2 : A Formal Data Model for IDS Alert Correlation," in *Proceedings of the 5th Symposium on Recent Advances in Intrusion Detection*, Zurich, Switzerland, October 2002.

[7] R. Ritchey, B. O'Berry, S. Noel, "Representing TCP/IP Connectivity for Topological Analysis of Network Security," in *Proceedings of 18th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2002.

[8] S. Noel, S. Jajodia, B. O'Berry, M. Jacobs, "Efficient Minimum-Cost Network Hardening via Exploit Dependency Graphs," *Proceedings of 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2003.

[9] S. Jajodia, S. Noel, B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Kluwer Academic Publisher, 2004.

[10] R. Ritchey, P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, 2000.

[11] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated Generation and Analysis of Attack Graphs," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, 2002.

[12] L. Swiler, C. Phillips, D. Ellis, S. Chakerian, "Computer-Attack Graph Generation Tool," in *Proceedings of DARPA Information Survivability Conference & Exposition II*, June 2001.

[13] J. Dawkins, C. Campbell, J. Hale, "Modeling Network Attacks: Extending the Attack Tree Paradigm," in *Proceedings of Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Johns Hopkins University, June 2002.

[14] P. Ammann, D. Wijesekera, S. Kaushik, "Scalable, Graph-Based Network Vulnerability Analysis," in *Proceedings of 9th ACM Conference on Computer and Communications Security (CCS)*, Washington, DC, November 2002.

[15] S. Noel, S. Jajodia, "Managing Attack Graph Complexity through Visual Hierarchical Aggregation," in *Proceedings of the ACM CCS Workshop on Visualization and Data Mining for Computer Security*, Fairfax, Virginia, 2004.

Part III. List of Publications

Authored Book

1. Peng Ning, Sushil Jajodia, X. Sean Wang, Intrusion Detection in Distributed Systems: An Abstraction-based Approach, ISBN 1-4020-7624-X, Kluwer Academic Publishers, Boston, 2003, 156 pages.

Edited Book

1. Daniel Barbara, Sushil Jajodia, Applications of Data Mining in Computer Security, ISBN 1-4020-7054-3, Kluwer Academic Publishers, Boston, 2002, 252 pages.

Research Publications

1. Lingyu Wang, Anyi Liu, Sushil Jajodia, "An efficient and unified approach to correlating, hypothesizing, and predicting network intrusion alerts," *Proc. 10th European Symposium on Research in Computer Security (ESORICS), Springer Lecture Notes in Computer Science, Vol. 3679,* Sabrina De Capitani di Vimercati, Paul Syverson, Dieter Gollmann, eds., Milan, Italy, September 2005, pages 247266 (Acceptance ratio 26/158).

2. Xinyuan Wang, Shiping Chen, Sushil Jajodia, "Tracking anonymous peer-to-peer VoIP calls on the Internet," *Proc. 12th ACM Conf. on Computer and Communications Security (CCS),* Alexandria, Virginia, November 2005, pages 81-91 (Acceptance ratio 38/250).

3. Steve Noel, Sushil Jajodia, "Understanding complex network attack graphs through clustered adjacency matrices," *Proc. 21st Annual Computer Security Conference (ACSAC),* Tucson, AZ, December 5-9, 2005, pages 160-169.

4. Vipin Swarup, Sushil Jajodia, Joseph Pamula, "Rule-based topological vulnerability analysis," Proc. 3rd Int'l. Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS 2005), Springer Lecture Notes in Computer Science, Vol. 3685, Vladimir Gorodetsky, Igor Kotenko, Victor Skormin, eds., St. Petersburg, Russia, September 24-28, 2005, pages 23-37.

5. Steven Noel, Michael Jacobs, Pramod Kalapa, Sushil Jajodia, "Multiple coordinated views for network attack graphs," IEEE Workshop on Visualization for Computer Security (VizSEC2005), Minneapolis, MN, October, 2005, pages 99-106.

6. Steve Noel, Sushil Jajodia, Eric Robertson, "Correlating intrusion events and building attack scenarios through attack graph distances," *Proc. 20th Annual Computer Security*

Applications Conference, Tucson, Arizona, December 6-10, 2004, pages 350-359.

7. Steve Noel, Sushil Jajodia, "Managing attack graph complexity through visual hierarchical aggregation" *Proc. ACM Workshop on Visualization and Data Mining for Computer Security,* October 2004, pages109-118 (Acceptance ratio 13/36).

8. Kaushal Sarda, Duminda Wijesekera, Sushil Jajodia "Implementing consistency checking in correlating attacks," Proc. First International Conference on Distributed Computing and Internet Technology, Springer Lecture Notes in Computer Science, Vol. 3347 (R. K. Ghosh and H. Mohanty, eds.), 2004, pages 379-384.

9. Steve Noel, Sushil Jajodia, Brian O'Berry, Mike Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," Proc. 19th Annual Computer Security Applications Conference, Las Vegas, Nevada, December 8-12, 2003, pages 86-95.

10. Daniel Barbará, Yi Li, Jia-Ling Lin, Sushil Jajodia, Julia Couto, "Bootstrapping a data mining intrusion detection system," *Proc. ACM Symp. on Applied Computing (SAC),* Melbourne, FL, March 2003, pages 421-425.

11. Yingjiu Li, Ningning Wu, Sushil Jajodia, X. Sean Wang, ``Enhancing profiles for anomaly detection using time granularities," *Jour. of Computer Security,* Vol. 10, No. 1/2, 2002, pages 137-157.

12. Peng Ning, Sushil Jajodia, X. Sean Wang, ``Design and implementation of a decentralized prototype system for detecting distributed attacks," *Computer Communications,* Vol. 25, No. 15, September 2002, pages 1374-1391.

13. Peng Ning, Sushil Jajodia, Xiaoyang Sean Wang, ``Abstraction-based intrusion detection in distributed environments," *ACM Trans. on Information and System Security,* Vol. 4, No. 4, November 2001, pages 407-452.

14. Daniel Barbara, Ningning Wu, Sushil Jajodia, ``Detecting novel network intrusions using bayes estimators," *Proc. 1st SIAM International Conference on Data Mining (SDM 2001),,* Chicago, IL, April 2001.

15. Daniel Barbara, Julia Couto, Sushil Jajodia, Leonard Popyack, Ningning Wu, ``ADAM: Detecting intrusions by data mining," *Proc. IEEE Workshop on Information Assurance and Security,* West Point, NY, June 2001, pages 11-16.

16. Daniel Barbara, Julia Couto, Sushil Jajodia, Leonard Popyack, Ningning Wu, ``ADAM: Detecting intrusions by data mining," *Proc. IEEE Workshop on Information Assurance and Security,* West Point, NY, June 2001, pages 11-16.

17. Yingjiu Li, Ningning Wu, Sushil Jajodia, X. Sean Wang, ``Enhancing profiles for anomaly detection using time granularities," *Proc. 1st Workshop on Intrusion Detection Systems,* Athens, Greece, November 2000.

18. Jiahai Yang, Peng Ning, X. Sean Wang, Sushil Jajodia, ``CARDS: A distributed system for detecting coordinated attacks," in *Information Security For Global Information Infrastructures: IFIP TC11 Sixteenth Annual Working Conference on Information Security,* (Sihan Qing and Jan H.P. Elof eds.), Kluwer, Boston, August 2000, pages 171-180 (Acceptance ratio 50/180).

Book Chapters

1. Peng Ning, Sushil Jajodia, "Intrusion Detection Systems Basics," in *Handbook of Information Security,* Hossein Bidgoli, ed., John Wiley, 2004.

2. Peng Ning, Sushil Jajodia, "Intrusion Detection Techniques," in *The Internet Encyclopedia,* Hossein Bidgoli, ed., John Wiley, ISBN 0-471-22201-1, December 2003.

3. Daniel Barbara, Julia Couto, Sushil Jajodia, Ningning Wu, ``An architecture for anomaly detection," in *Applications of Data Mining in Computer Security,* Daniel Barbara, Sushil Jajodia, eds., ISBN 1-4020-7054-3, Kluwer Academic Publishers, Boston, 2002, pages 63-76.

4. Yingjiu Li, Ninging Wu, X. Sean Wang, and Sushil Jajodia, ``Enhancing profiles for anomaly detection using time granularities," in *Intrusion Detection,* Deborah Frincke, ed., IOS Press, Amsterdam, 2002, pages 137-157.